LASER POWER RESEARCH

P. 214

FINAL REPORT

Digital Image Profilers For Detecting Faint Sources Which Have Bright Companions NAS7-1103

May 25, 1992

Submitted to:
NASA/JPL
4800 Oak Grove Drive
Pasadena, California 91109

Submitted by:
Laser Power Research
12777 High Bluff Drive
San Diego, California 92130

PROJECT SUMMARY

The long-term objective of this program is the development of image profiling systems capable of detecting extremely faint optical sources which lie in close proximity to bright companion sources. Toward this objective, Phase I activities included the analysis of statistical profiling techniques, the construction and testing of a single-channel breadboard image profiler, together with the preliminary design of a multi-channel brassboard instrument.

The approach employed for detecting faint stellar companions is novel in three respects. First, it does not require an optical system wherein extraordinary measures must be taken to minimize diffraction and scatter. Second, it does not require detectors possessing either extreme uniformity in sensitivity or extreme temporal stability. Finally, the system can readily be calibrated, or nulled, in space by testing against an unresolved singular stellar source.

Experimental data derived from laboratory testing of the Phase I breadboard indicate that image profilers can exhibit a performance which approaches the theoretical limit imposed by photon statistics. Specifically, when used in conjunction with a space based telescope in the 2-3 meter class, a multi-channel version of the system should permit the detection of 17th and 19th magnitude stellar objects having angular displacements of 0.04 and 0.2 arc seconds, respectively, from a 6th magnitude object.

The Phase II program encompassed three principle activities; these being fabrication of a multi-channel image profiling system, laboratory testing of the system in the presence of simulated atmospheric turbulence and field testing of the system on a 24-inch telescope at Table Mountain. By incorporating automatic image enhancement within the laboratory prototype, it was demonstrated that image degradation associated with atmospheric turbulence can, in large measure, be eliminated via the image profiling technique. Although adverse weather restricted subsequent field testing to a few nights only, it was shown that the operating characteristics of the multi-channel system conform closely with theoretical predictions.

As evidenced from the successful Phase II activities, we find that the image profiling technique is directly applicable to resolution enhancement of ground based astronomical telescopes. It also has significant potential in precision spectral analysis and in the detection of trace impurities via spectrophotometric techniques. As such, instrument designs emerging from the program offer a significant commercial market.

THIS PAGE INTENTIONALLY LEFT BLANK

LASER POWER RESEARCH

FINAL REPORT

Digital Image Profilers For Detecting Faint Sources Which Have Bright Companions NAS7-1103

May 25, 1992

Submitted to:
NASA/JPL
4800 Oak Grove Drive
Pasadena, California 91109

Submitted by:
Laser Power Research
12777 High Bluff Drive
San Diego, California 92130

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF APPENDICES

LIST OF FIGURES

LIST OF FIGURES (CONTINUED)

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The long-term objective of this program is the development of image profiling systems capable of detecting extremely faint optical sources which lie in close proximity to bright companion sources. Toward this objective, Phase I activities included the analysis of statistical profiling techniques, the design, construction and testing of a single-channel breadboard image profiler, together with the preliminary design of a multichannel brassboard instrument. Building and testing of the brassboard instrument were completed during the Phase II program. The ensuing report on the overall activity is divided among five principal topics; these being entitled Introduction to image profiling techniques, Review of the Phase I program, Design and fabrication of the multichannel system, Laboratory testing, and Field test program.

I. INTRODUCTION TO THE IMAGE PROFILING TECHNIQUE

To detect a faint optical source that is located in close proximity to a much brighter source, it is necessary to develop a detection system which is capable of distinguishing very small fractional variations in radiation flux. This requirement is unavoidable insofar as the faint source invariably appears superimposed upon a background which includes radiation from the brighter source; some of which has been scattered by imperfections in the imaging system; some of which is an intrinsic part of the point spread function of the brighter source. Unfortunately, conventional detection processes are unsuited to the sensing of small fluctuations superimposed upon a strong background. Basically, optical image detectors measure the total flux at each point in an image, rather than small variations in intensity relative to adjacent points in the image. Thus, the fundamental limitation associated with the detection of a faint source superimposed upon a bright background lies in the statistical uncertainty of the background. When detection is performed by means of photoemissive detectors, this statistical uncertainty applies to the photoelectrons which comprise the background.

The image profiling system developed during the course of this program potentially can reach the detection limit imposed by the quantized nature of photoelectron statistics. To accomplish this, the system performs many scans of the image area; storing the photoelectron counts from each element of the image in digital form. Furthermore, recognizing that angular jitter may exist in the optical system line-of-sight, the digitized data from each scan are recentered prior to being added to the data accumulated from previous scans. In this manner, a digitized image is built up in a manner wherein the effective jitter is very small compared to the diffraction limit of the optical system. Also, independent of the number of detectors employed, each detector scans the entire image during the course of every scan. This time-sharing technique circumvents the need for extremely uniform detectors.

A simplified one-dimensional slit scanning and digital processing system is shown schematically in Figure 1. In operation, the composite bright/faint image is scanned repetitively across a single slit. The slit width is on the order of one-tenth the diameter of the central bright zone of a diffraction limited Airy disc. Behind the slit is a single photomultiplier tube. The amplified and conditioned output pulses from the photomultiplier are entered into a dynamic shift register. Whenever a pulse is received from the preamplifier, the next clock pulse enters a "one" into the register; otherwise, each clock pulse enters a "zero" into the register. In this manner, the pulse density profile of the scanned image is impressed into the shift register in much the same way that a string of digital data can be recorded on a magnetic tape. A shift register, however, has a distinct advantage over a tape, in that a shift register can be stopped or started within one clock pulse.

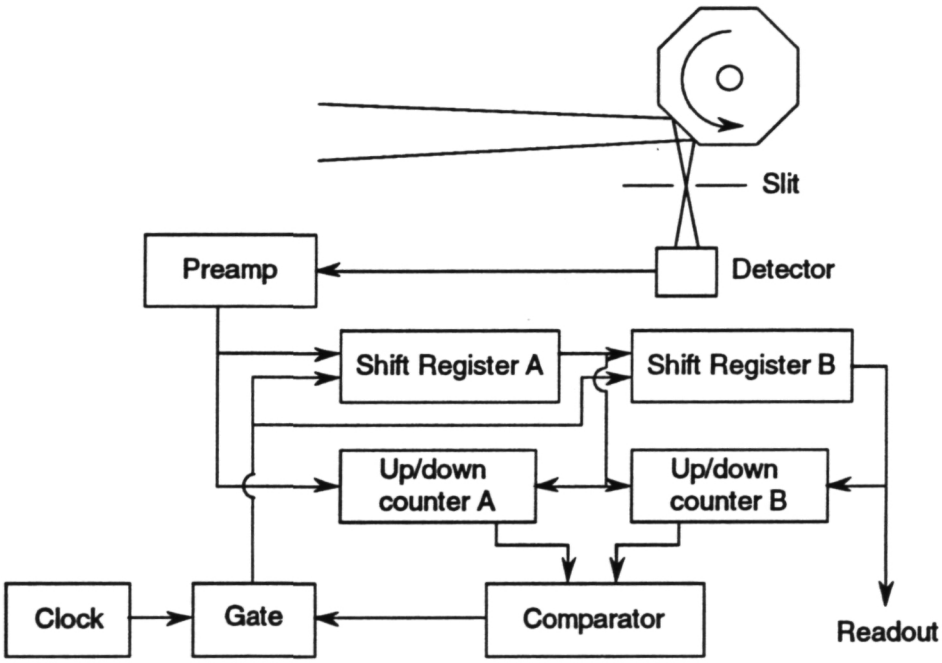


Figure 1. Simple slit scanning and digital preprocessing system

Let us examine the behavior of this simple system when we scan the image of a single bright source across the slit. Here, it is assumed that the scanning mirror provides a reference pulse such that the shift registers are reset and the counters are zeroed shortly before the image begins its traverse of the slit. Thus, during the arrival time of the pulse train, counter A develops a count which equals the total number of photoelectrons derived from the scan. The traveling count stored in counter A remains constant until the scan induced pulse train begins to exit shift register A and to enter shift register B. At this point, the count in register A starts to decrease, while the count in register B begins to climb. At the instant when the count in shift register A is identical to that in shift register B, the comparator inhibits further clocking of both registers. Thus, the shift registers

are stopped such that the pulse train (which represents the intensity profile of the scan) is centered precisely within the registers; i.e., the image profile received during the scan is centered perfectly within the limits imposed by the statistics of a quantized process.

We now must extract the "frozen" image profile from the shift register such that it can be added to the profiles obtained from previous scans. If, for example, we wish to divide the scan into 100 increments, we must clock the register so as to accumulate a separate count from each one-percent of the register. The resulting 100 separate counts then are added to the corresponding sums of counts from all previous scans.

Having performed a calibration run against a non-resolved singular source, together with a similar run against an unknown source, the detection of faint sources in the vicinity of the latter is accomplished by a normalization and subtraction process. First, the incremental counts which comprise the image profile of the unknown source are normalized such that the total number of counts within the image matches the number of counts in the calibration image. Each incremental count associated with the calibration profile is then subtracted from the corresponding incremental count of the unknown profile. The differential counts are then plotted to reveal any difference between the profiles. If the profile of the unknown includes a faint source which is displaced with respect to the primary source, the relative intensity and position of the faint source become apparent in the differential plot.

A single-channel system is inefficient in that it falls far short of using all the photons available within the image. For example, a system which divides an image profile into a hundred increments typically must employ a slit width which encompasses no more than one percent of the scan. Thus, it collects no more than one percent of the available photons and, as a consequence, the achievement of any specified statistical precision requires an integration time which is one-hundred times greater than that required by an ideal system.

To more closely approach the efficiency of an ideal system, it is necessary to use a multichannel approach. A multichannel profiling system constitutes a simple expansion of the single-channel system. As such, it incorporates an array of slit-like apertures which lay side by side, so that the scanned image passes over each aperture in succession. (Feeding multiple detectors from a close-packed multi-slit focal plane array can be accomplished by fiber optics.) Each aperture is complemented by a photomultiplier, a preamplifier, a shift register, and a single-channel incremental count accumulator. Thus, the multiple channels operate in parallel to the point where the counts from individual accumulators are summed in a multichannel system accumulator.

As noted earlier, the ability of an ideal profiling system to detect a faint source is set by the statistics of the photon counts which make up the background in the immediate vicinity of the source. Specifically, the uncertainty, or *one-sigma*, in measuring the intensity of a single increment of the scanned field is given by the square root of the mean number of counts within the increment. The extent to which the performance of a real system approaches that of its ideal counterpart depends upon the nature and magnitude of other error sources. In the case of an image profiling system, other sources of error fall into two distinct classes; those which are constant with time and those which vary with time.

Systematic errors which are constant with time can be dealt with by a simple calibration procedure. This fortuitous situation arises because, independent of the nature of such errors, all unresolved singular sources of approximately equal brightness and spectral distribution produce identical image profiles. This is not to say that their profiles are of identical amplitude; but, when normalized to a common amplitude, the shapes of their profiles are identical to within the statistical uncertainty defined above. Thus, by conducting calibration runs with respect to sources which are known to be singular, a point-by-point reference profile can be established. When this reference profile is compared to the profile of an unknown source, any difference between the profiles which is in excess of that associated with predictable statistical errors, must arise either from the presence of one or more additional sources, or from systematic errors which vary with time.

As an outgrowth of the above argument, we arrive at the key advantage that the digital profiling approach possesses with respect to conventional imaging techniques. Specifically, the precision achievable with a digital profiling technique is dependent only upon the fundamental statistics of the photon detection process and upon the *variability* of the systematic errors. If the variability of the systematic errors can be maintained at a negligible level for the period of time required to perform a calibration/measurement run, then system performance is equivalent to that of a perfect system; i.e., the precision achieved is dependent only upon the fundamental statistical process. Fixed systematic errors of reasonable magnitude, such as those associated with scatter and diffraction in the optical train, non-linearity of the scanning mechanism, variations in sensitivity from one detector to another, etc., have little or no effect upon performance.

At this point in our discussion, it becomes instructive to establish a ballpark performance estimate for an ideal multichannel image profiler when used in conjunction with the Hubble Space Telescope. For this purpose, let us consider as the bright source a type G star with a visual magnitude of 6. (Magnitude 6 has been chosen arbitrarily, in that there are about 5,000 stars of this magnitude or brighter.) Using a spectral bandwidth extending from 350 to 500 nanometers, we find that the useful flux entering the telescope from such a source amounts to about 1.8×10^8

photons per second. If we assume that the product of the telescope transmission and the photomultiplier quantum efficiency is 0.1, the count rate created in a fixed detector encompassing the full field of view becomes 1.8×10^7 photoelectrons per second. If the image of the light source encompasses ten scan elements (i.e., about 10-percent of a 100-element profile), then the rate at which counts are accumulated in the central elements of the scan is on the order of 1.8×10^6 per second. Recognizing that a mean count of 10^8 is needed to achieve a primary image profile precision of 10^{-4}, we find that this degree of precision requires an integration time of about 56 seconds.

In relation to the Space Telescope, the above precision allows the detection of a secondary source having approximately 10^{-4} times the brightness of the primary source, provided the angular displacement of the source is on the order of 0.04 arc second, or greater. Further, as the angular displacement is increased to about 0.2 arc second, the relative intensity of a detectable secondary falls to the 10^{-5} regime. Converting to visual magnitudes, we find that an ideal space-based system with an integration time on the order of one minute is capable of detecting a 16^{th} magnitude secondary at a displacement of 0.04 arc second from a 6^{th} magnitude primary. Likewise, such a system can detect a secondary of magnitude 18.5 at a displacement of 0.2 arc second from a 6^{th} magnitude primary.

The above conclusions apply to the performance of an image profiler when operated in conjunction with a space-based telescope. However, system tests conducted with the Phase II multichannel brassboard also included ground based operation from Table Mountain. It was therefore necessary to design the system so as to be capable of operation in the presence of a turbulent atmosphere. Toward this objective, we examined both the fundamental nature of atmospheric turbulence effects and the extent to which an image profiler can overcome such effects.

Turbulence within the atmosphere introduces optical wavefront aberrations which typically vary on a time-scale of 0.002–0.1 seconds. If we have a means for freezing the instantaneous wavefront on a shorter time-scale, we find that the aberrations can be described as a superposition of simple trigonometric terms. This is referred to as Zernicke polynomial analysis. When the aberrations introduced by turbulence are described in this manner, we find that the relative magnitude of various terms can be expressed as a power spectrum. Here, we assume that the RMS value of each term in the polynomial is derived from a large number of frames which are accumulated in the presence of a statistically stationary turbulent process.

By characterizing an aberrated wavefront in the above fashion, the terms of greatest amplitude are those with the lowest spatial frequency; terms which correspond to tip and tilt of the wavefront and which cause the image of a point source to dance around in the focal plane of an optical instrument.

Thus, relative to these specific components of aberration, we can use a short exposure time to create a high-resolution image.

Relative to tip and tilt contributions to wavefront aberration, the image profiling concept embodies an image recording process wherein these effects are eliminated. Specifically, by scanning the image plane with a slit, we cause the system to be insensitive to the instantaneous image displacement in one dimension. By electronically centering the resulting one-dimensional image profile, we achieve insensitivity in the orthogonal dimension. Since a complete scan of the image plane is accomplished within a few milliseconds, the effects of image motion during the scan are negligible.

As was shown experimentally, elimination of the effects of tip and tilt results in considerable resolution enhancement. However, the image profiling concept also addresses aberrations which are of higher order than tip and tilt. The next most serious contributors to wavefront aberration are those associated with focus. We refer to such aberrations in the plural insofar as the focusing effects of the atmosphere can be described by a pair of orthogonally oriented cylindrical lenses; the instantaneous optical powers of which are uncorrelated with respect to each other.

We already have noted that the instantaneous position of an image relative to a scanning slit is of no consequence to image quality along the axis which lies parallel to the slit. Likewise, instantaneous defocus of an image along the axis of a slit has no influence upon the quality of an image profile. Consequently, an intrinsic property of the profiling approach lies in its immunity to three of the four principal contributions to poor resolution in conventional imaging systems; i.e., immunity to tip, tilt, and defocus in one axis.

The effect of the remaining defocus term is addressed by integrating only those image frames wherein the magnitude of this term is minimal. In particular, we can use real-time electronic image quality sensing to reject all frames for which the image "sharpness" falls below a predetermined limit.

Strictly speaking, the above description of frame selection is an over-simplification. In practice, the image quality selection process does not differentiate the effects of defocus along the scan axis from the effects of higher order aberrations along the same axis. Rather, we use a collective criterion (referred to as the Strehl intensity ratio) to select superior frames. Thus, relative to aberrations along the scan axis, the technique singles out individual scans for which the ensemble effect of all aberrations falls below a predetermined limit. Establishing an appropriate Strehl intensity ratio limit can be performed automatically according to prevailing seeing conditions. Under conditions of typical seeing, the limit is adjusted so as to select a few percent of all scans.

II. REVIEW OF THE PHASE I PROGRAM

Technical objectives of the Phase I program encompassed the design and construction of a single-channel image profiler, the analysis of multichannel system performance, the testing of the single-channel profiler and the fast Fourier filtering of representative image profiles. These objectives were met successfully in the manner described below.

1. Design and Construction of a Single-Channel Image Profiler

To provide experimental verification of the digital image profiling concept, a single-channel profiler was breadboarded and tested under conditions simulating those at the focal plane of a large telescope. The breadboard comprised a multisource optical scanning subsystem and a signal processing subsystem; the design and operating characteristics of which are described below.

Multisource optical scanning subsystem: Prior to the commencement of Phase I, the concept for experimental testing of an image profiler had progressed to the approach shown schematically in Figure 2. The strawman characteristics of the test system were those listed in Table I. Subsequent to contract award, several features of the original concept were refined. The most significant modification involved the adoption of a scanning multi-lens assembly in place of the scanning octagonal mirror. This alternate geometry allowed the scanning assembly to be mounted upon a precision phonograph turntable; thereby taking advantage of the constant-speed low-RPM motor technology intrinsic to the audio reproduction arena. By driving a nine-lens scanner at 33 1/3 RPM, the originally proposed frame rate of 5 hertz was preserved.

Table I. Phase I test system parameters

Scanning system	9-facet mirror equivalent
Scanner rotation rate	33 RPM
Slit width	0.05 cm
Slit length	5.0 cm
Photomultiplier response	S-20
Frame rate	5 Hz
Scan duration	0.1 sec
Readout time	0.1 sec
Shift register capacity	1.025×10^6 counts
Clock rate	10.24 MHz
Up/down counter capacity	10^6 counts

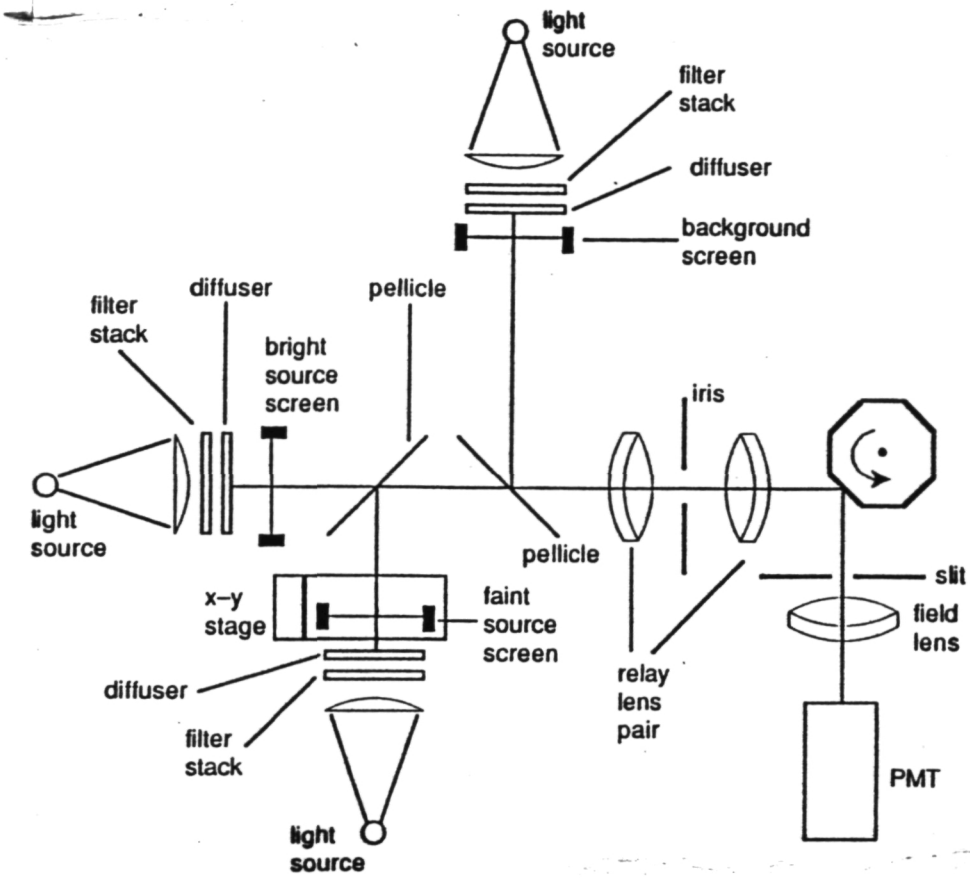


Figure 2. Multisource test system

Other modifications to the original concept involved a change in spectral response from S-20 to S-11, the deletion of a separate background source, and the replacement of the focal plane slit by its fiber optic equivalent. The switch in spectral response was made for two reasons. First, it was recognized that the enhanced short wavelength quantum efficiency of the S-11 response would provide superior performance with respect to most of the astronomical objects of interest during the Phase II program. Second, there became available an extremely compact S-11 tube which had been developed specifically for high-speed counting of single photons. The need for a separate background source was eliminated when computer generated digital mask technology was adopted for simulation of the primary and secondary sources. Using this technology, it became easier to introduce the desired background via the primary mask than to sustain the increased complexity and cost of a third source. Replacement of the single slit by a fiber optic bundle also was prompted by the anticipated needs of the Phase II program. In this case, it was recognized that the fiber optic approach becomes a necessity in a multichannel design. Incorporation of these features led to the breadboard subsystem design shown in Figure 3.

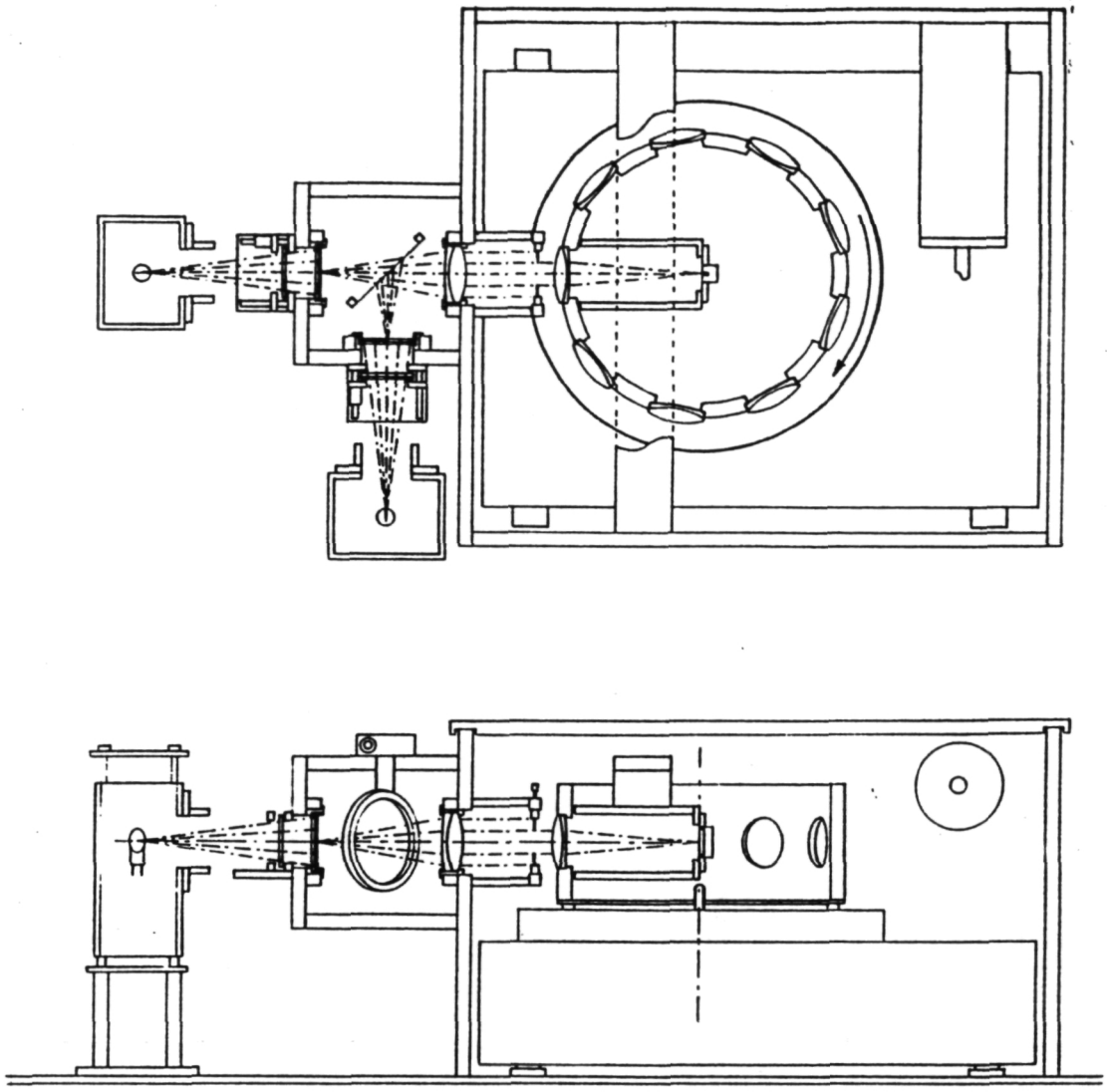


Figure 3. Breadboard image profiling test system

On the left-hand side of Figure 3 is shown the dual source image projector. Each image source comprises a stable tungsten halogen lamp, a filter stack, a diffuser and a source screen. The filter stack can incorporate both spectral bandpass filters and neutral density filters; thereby providing independent control over the spectral content and brightness of each image. The source screens employed for most of the experimental work are computer generated digital equivalents of a highly magnified Airy disc. The large-scale negative master of a typical screen is shown in Figure 4. The geometric scale of the positive screens used in each source is adjusted so as to allow projection at unity magnification onto the focal plane of the image profiler. Superposition of bright and faint sources is accomplished by means of an uncoated pellicle. To allow the spatial coordinates of the faint source to be adjusted with respect to those of the bright source, the combining pellicle is

mounted upon a precision rotary stage. After collimation via a coated achromat, the combined beam passes through the iris diaphragm which provides intensity control of the composite image.

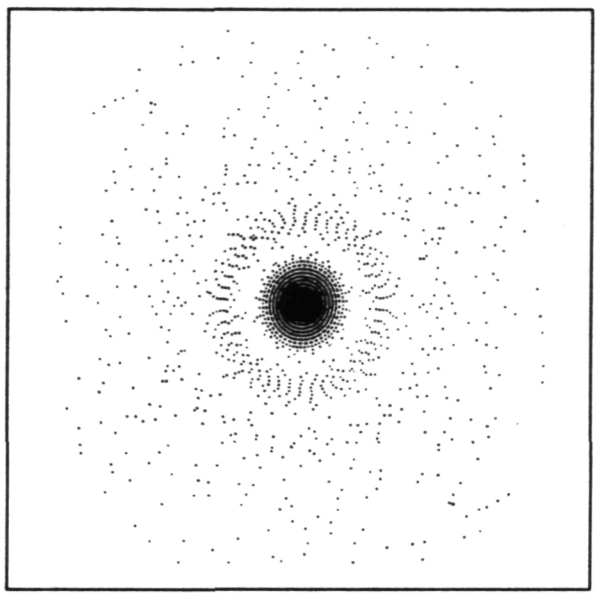


Figure 4. Digitized $(\sin x)^2/x^2$ intensity distribution

The composite beam is intercepted by the rotating multi-lens scanner. The nine achromats incorporated within the scanner have back focal distances which coincide with their radial displacement from the rotation axis of the scanner. This geometry causes an image of the composite source to be scanned across the rotation axis at a frame rate of 5 hertz when the scanning assembly is rotated at 33 1/3 RPM. The Denon turntable selected to drive the assembly employs an integral magnetic disc and tape head pickoff to provide closed-loop angular rate control with respect to a crystal oscillator. The turntable also has a precise speed control whereby the rotation rate can be adjusted in discrete steps of ± 0.001. This feature provides a convenient means for investigating the effects of scan rate errors upon profiler performance.

The slit-like input end of a fiber optic bundle is located at the center of the multilens scanner; the long axis of the slit lying parallel to the rotation axis. The 0.5-millimeter width of the slit represents one-percent of the width of the scan and encompasses fourteen rows of fibers. The fibers within the bundle are randomized so as to provide an output distribution which has minimal dependence upon vertical displacements of the scanned image. The output of the fiber bundle illuminates the photocathode of a photomultiplier tube that is housed within the light-tight cylinder shown in the upper right-hand corner of Figure 3.

Signal processing subsystem: The breadboard signal processing system essentially is identical to that originally proposed for the Phase I program and, as such, its operating characteristics conform to those listed in Table I. The physical breakdown of the electronic hardware includes three distinct subunits; these being the I/O card internal to the computer, the control board and the shift register boards. Detailed signal processor schematics are presented in Appendix D.

The I/O card, which operates with an 8255 bus interface chip, is hardwired to addresses 0300 and 0303 within the IBM-PC. Consequently, it is via these addresses that the computer can transmit 8-bit bytes, receive 8-bit bytes, send control commands, and control the I/O card functions.

The control board encompasses the counters which synchronize the read and write clocks, the control latches, the centration counters, the clock logic, and the input signal conditioning circuitry.

Each shift register board contains one-quarter of the shift register capacity. The exact capacity of the register is dependent upon the number of bins into which each profile is divided. For the 128-bin arrangement employed throughout Phase I, the capacity is 1,015,808 bits. It should be noted that the number of bins into which the profile is divided is not hardwired into the system, but is software selectable in multiples of eight.

The supporting software routines are written entirely in Microsoft BASIC and compiled to offer the maximum advantage in speed. All raw counts, normalized counts and accumulated counts are stored in double precision. The basic modes of the software include system initialization, main program loop, and data storage/display; each of which is described briefly below.

During system initialization, the operator can select both the number of bins into which each count is subdivided and the number of scans which are integrated prior to generation of a difference profile. Upon initialization by the operator, the computer resets all clocks, determines the number of clock pulses which must offset the read and write clocks of the FIFO shift registers, and loads the number of clock pulses which will be used to read out the incremental data stream.

The main program loop commences by flushing the shift registers in parallel, such that all registers contain zeros. At this point, the operator is given the option of creating a new profile count file or of appending the forthcoming data to a previously created file. The system then awaits a scan initiation signal from the optical scan system. Upon scan initiation, the counters which determine the shift register content are activated. When the count at the input of the first half of the register is equal to twice the count at the input to the second half of the register, the clock pulses to the register are inhibited. The counts within the register are then read out in eight parallel data streams.

Thus, in the case of a 128-bin profile, the counts are accumulated from eight ports while the register is advanced through 16 sequences of 7,936 clock cycles.

In the data storage/display mode, the accumulated bin counts are normalized and displayed upon a monitor. The operator can select the frequency with which the display is updated. When each set of data is displayed, the operator has the option of discarding the set or of recording it to the data file. Two data files are maintained continuously. One contains the accumulated photon counts in each bin; the other contains the periodically updated and normalized counts. The latter counts are normalized with respect to the sum of accumulated counts in all bins.

2. Analysis of Multichannel System Performance

Throughout the Phase I program, a concurrent analytic effort was directed toward parametric refinement of the image profiling technique. This effort placed emphasis upon deriving of an optimum set of system parameters pertaining to the multichannel brassboard which was to be constructed during Phase II. An integral part of the analysis encompassed the development of computer modeling techniques whereby the influence of each system parameter could be assessed. Optical characteristics of importance in this regard included the choice of spectral bandwidth, the spectral response of detectors, the number of elements encompassed by the scan width, the scan frame rate, and the number of independent channels. Electrical characteristics included prescaler divide ratio, shift register clock rate, shift register capacity, and duty cycle. Our discussion of these analytic efforts is, for convenience, divided among three topics; namely Image profiling by computer simulation, Fast Fourier transform filtering and Prescaler statistics. More detailed discussions of each topic can be found in Appendices A, B and C.

Image profiling by computer simulation*: Generation of profile data via slit scanning of a real image occurred toward the end of the Phase I program. Thus, to examine signal processing techniques at an early stage of the program, it was necessary to develop a computer model for the simulation of such data.

To provide a realistic simulation, the computer model employs a full Bessel function Airy disc superposition of a primary and a secondary source. The parameters describing the composite source include the intensity of the primary source (given in terms of the number of photoelectrons accumulated within a central bin of a multiple scan), the relative brightness of the secondary source, and the angular separation of the secondary source with the respect to the primary. The parameters describing the scan system include the width of the scan, the height of the scanning slit,

* See Appendix A for full mathematical treatment

the number of bins into which the profile is subdivided, and the fractional scanning speed error which occurs relative to the profile of a reference source. To preserve scalability to any optical system, the geometric parameters describing scan width, slit height and primary/secondary separation are written in units of the diffraction limited angle, λ/D.

Features of the program include the decentration of a primary centroid to account for the presence of a faint secondary, the introduction of normally distributed noise to each bin count, and the normalization of a reference profile with respect to the composite profile of a primary and secondary source. The output of the program comprises a normalized difference signal analogous to that produced by the breadboard image profiler.

To illustrate the characteristics of the program, it is convenient to examine a typical example of a synthesized difference profile. The profile shown in Figure 5 corresponds to the input conditions listed in Table II.

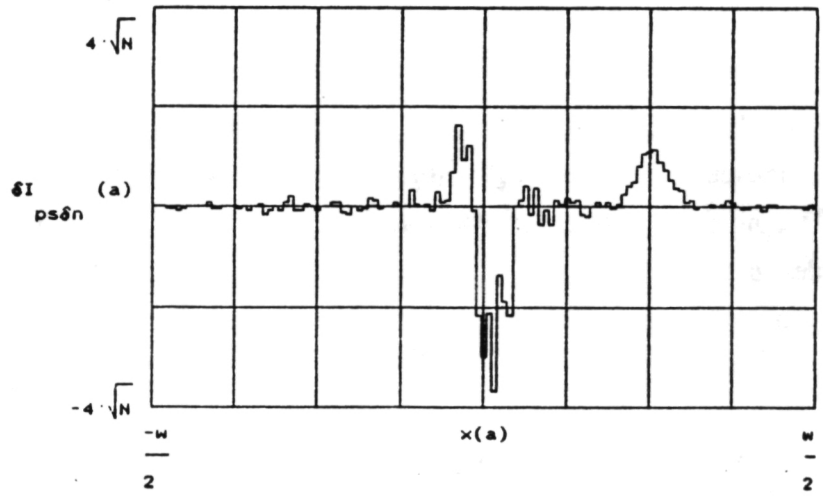


Figure 5. Unprocessed noisy difference profile

Table II. Input conditions

Scan width	(w)	16λ/D
Slit height	(h)	32λ/D
Number of bins	(M)	128
Fractional scan error	(δf)	0.0001
Maximum bin count	(N)	10^8
Secondary intensity	(β)	0.0001
Secondary displacement	(θ)	4λ/D

Examination of Figure 5 reveals several significant features. First, the presence of a secondary source having an amplitude of about $\sqrt{N}$ and a displacement of about $4\lambda/D$ is clearly evident. It

should be noted that the vertical scale of the plot extends from $-4\sqrt{N}$ to $+4\sqrt{N}$; i.e., between the limits of ±4-sigma relative to the largest bin counts. Thus, in terms of primary source intensity, the scale extends from -0.0004 to +0.0004.

The feature at the center of the scan is somewhat more complex; showing an increased noise level, together with a strong minimum to the right of center and a subsidiary maximum to the left of center. The increased noise level reflects the larger photoelectron counts accumulated within the central bins of the profile. The strong minimum occurs because the reference profile has been normalized such that its total count equals the sum of the counts encompassed within the unknown profile. Consequently, in the vicinity of scan center, the normalized intensity of the reference is greater than that of the unknown profile. Thus, when the reference profile is subtracted from the unknown profile, a minimum occurs in the vicinity of the scan center. Because the primary is offset from center due to the presence of a secondary, the minimum is displaced slightly to the right of center. Meanwhile, to the left of center, the displaced primary is slightly more intense than the reference; thereby causing a subsidiary maximum. In the absence of noise, the areas which represent the secondary source and the arithmetic sum of the central maximum/minimum are equal and opposite.

Extensive use of the modeling program provided invaluable insight into the characteristics of profiling systems. In particular, its use greatly simplified the task of developing the processor control algorithms. During the latter part of Phase I, the program was used to provide synthetic data streams for the design and optimization of the Fourier transform filtering process described in the ensuing subsection. It also provided a large fraction of the material used to develop the multichannel system design.

Fast Fourier transform filtering*: The Phase I program produced a breadboard image profiling system which generates a profile in the form of discrete bin counts. The noise associated with each bin count is consistent with a normal distribution wherein the *one-sigma* noise amplitude for a bin is equal to the square root of the mean number of counts in that bin. If the counts within adjacent bins are uncorrelated, this noise level establishes the limit of sensitivity for the system. However, since the profile of an image extends to several bins, there exists a degree of correlation between neighboring bin counts. This interbin correlation can be used to improve the final signal-to-noise ratio by a variety of spatial filtering and/or profile smoothing techniques. Within the scope of Phase I, several such techniques were examined; the most successful being that of fast Fourier transform filtering.

---

* See Appendix B for full mathematical treatment

The most sophisticated signal processing program developed during Phase I provides raw data filtering via a four-stage process. First, the high-frequency noise is removed from the normalized difference signal. Second, the reference profile is corrected to first order for the presence of a suspected companion source. Third, the refiltered signal is corrected for scan rate errors. Finally, the corrected difference signal is analyzed to determine the relative intensity and position of companion sources.

For typical profiles, application of the above described process enhances the signal-to-noise ratio by a significant factor. Using the raw data profile presented in Figure 5, for example, the first three stages of the filtering process provide the deterministically enhanced signal profile presented in Figure 6. From an examination of many filtered profiles of this type, it has been concluded that the resultant signal-to-noise ratio enhancement factor lies between four and six. In the context of detecting faint stellar companions, such an enhancement is equivalent to lowering the companion detection threshold by some 1.5 to 2.0 astronomical magnitudes.

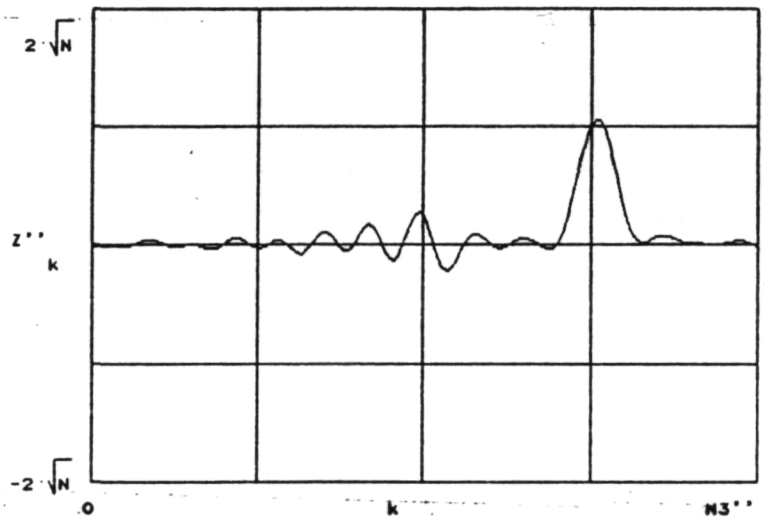


Figure 6. Fully corrected and filtered profile

Prescaler statistics*: In prior discussions of signal processing, it has been assumed that the incoming photoelectron stream is converted into a pulse sequence which is clocked directly from the preamplifier into a shift register. This also was the approach taken experimentally throughout Phase I. However, during Phase I, it became increasingly apparent that the introduction of a prescaler between the preamplifier and the shift register offers advantages both in performance and cost. (A prescaler performs pulse rate division by forwarding a single pulse to the shift register at the conclusion of each period within which a specified number of pulses are received from the

* See Appendix C for full mathematical treatment

preamplifier.) Thus, within the scope of our Phase I analytic studies, we developed a computer model of a pulse processing system which includes a prescaler. The advantages of a prescaler are threefold. First, it provides an upward expansion of the dynamic range of the profiling system. Second, it reduces electronic bandwidth requirements for most elements of the signal processor. Third, and perhaps most importantly, it greatly reduces the required capacity of the shift registers; thereby effecting a significant cost reduction in a multichannel system.

The expansion in dynamic range is best understood via an examination of the dead time coefficient which is intrinsic to any system which counts random events. When the time of occurrence for a sequence of events is random, the probability that an event occurs within any specified interval becomes the product of the interval duration and the mean rate of occurrence. Thus, no matter how short an interval is chosen, the probability that an event occurs within that interval is finite. All real systems exhibit time intervals within which it is impossible to distinguish between one event and two events. This interval is characteristic of the specific system and is referred to as the system *dead time*. For situations wherein the mean time between events is large compared to the dead time, the fraction of counts which fail to be registered as a consequence of this phenomenon remains small. It is, therefore, a simple matter to apply a small count correction factor which is based upon the measured value of system dead time. However, as the mean time between events is reduced and the magnitude of the correction increases, the accuracy with which the real count can be determined becomes degraded. In the context of an image profiling system, this degradation in accuracy sets an upper bound upon the count rate which can be accommodated; i.e., it imposes an upper limit to the system dynamic range.

In the absence of a prescaler, the dead time of an image profiler can be no less than the period between shift register clock pulses. Throughout the Phase I program, we used a clock rate in the vicinity of 20 megahertz; yielding a theoretical dead time of 50 nanoseconds. However, the presence of waveform imperfections and logic pulse jitter caused the effective dead time to be somewhat longer; typically on the order of 60 nanoseconds. The maximum practical count rate consistent with this dead time is by no means clear cut. In particular, it depends strongly upon the accuracy which must be achieved and upon the effort one is prepared to expend in calibrating the system. However, for the purpose of both the analytic and the experimental activities of Phase I, we adopted a value of 10^5 counts per second as being desirable, while 10^6 counts per second was set as an upper limit.

It is instructive to examine the above limits in the context of an image profiling system located at the focal plane of the Hubble Space Telescope. For this purpose, we assume a profiling system which employs a 150-nanometer bandwidth centered at 425 nanometers and which exhibits an an-

gular resolution equivalent to one-tenth of the resolution of the telescope. Also, we assume that the shortest possible measurement time is desired; i.e., that incoming signal intensity must not be sacrificed by the insertion of an optical attenuator. Under these circumstances, the visual magnitude of a source corresponding to the 10^6 counts per second upper limit is approximately 6.5. For the more desirable rate of 10^5 counts per second, the brightest source is reduced to visual magnitude 9. In either case, it is evident that the intensity of many sources of interest exceeds the system dynamic range by a significant factor.

Within the present state of the art, it is impractical to increase dynamic range significantly by increasing the clock rate of shift registers. Also, recognizing that an increase in clock rate must be matched by an increase in shift register capacity, such an approach becomes prohibitive from a cost standpoint. On the other hand, prescalers with bandwidths of several hundred megahertz are readily available. The question which must be addressed, therefore, is "Does the use of a prescaler impair the statistical precision of an image profiler?"

A detailed analysis of the effects of prescaler divide ratio upon profiling precision has shown that, for typical photoelectron count rates, divide ratios up to 20:1 have no significant effect upon precision. Further, the use of a prescaler with a modest divide ratio of 10:1 allows a dramatic reduction in shift register clock rate and shift register capacity. Specifically, the inclusion of a divide-by-ten prescaler allows us to reduce the shift register clock speed from 20 megahertz to 500 kilohertz with no loss in performance. At first glance, it seems surprising that a ten-to-one reduction in count rate allows a forty-to-one reduction in clock speed. However, it must be recognized that, in contrast to the undivided photoelectron pulse stream, the output pulse train from a prescaler is far from random.

3. Testing of the Single-Channel Profiler

The principal experimental activities of Phase I encompassed the setting up and calibration of the single-channel breadboard system and the evaluation of profiler performance in the context of detecting faint companion sources. In addition, a lesser effort was directed toward Fourier transform filtering of representative signal profiles. Although not included within the original Statement of Work, this last activity was deemed worthwhile insofar as the results of the activity were expected to aid significantly in the subsequent design of a multichannel system. Results and comments pertaining to each of the above activities are presented below.

Setup and calibration: In preparation for multi-source profiling, the breadboard system was set up so as to provide single-source profiles consistent with those employed in the earlier analytic

work. Using a source screen of the type described, the attenuating filters and iris diaphragm of the multi-source test system were adjusted so as to provide approximately 1000 counts per scan into each of the central bins when the total number of bins was set at 128. Figure 7 illustrates a typical single source profile obtained in this fashion when using a screen with a radial distribution of the form $(\sin x)^2/x^2$. Note that the integration of 8000 scans provides the anticipated count of about 8×10^6 in each of the central bins. As expected, a comparison of the profile in Figure 7 with that derived for a J1 Bessel function* reveals that the $(\sin x)^2/x^2$ screen exhibits somewhat more intense first-ring maxima than those of the latter function. In the region of the second and third rings, however, the less well defined profile of the screen has an intensity similar to that of the J1 Bessel function. At the outer extremity of the profile, the intensity of the screen falls off more steeply than that of the theoretical function. However, from an experimental standpoint, it is important only that the real profile exhibit an intensity at the displacement of the secondary source which is no less than the intensity assumed for the purpose of theoretical modeling.

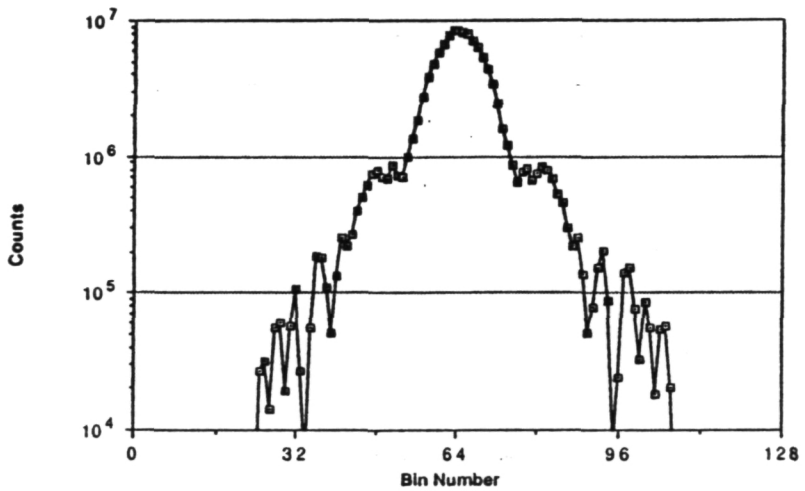


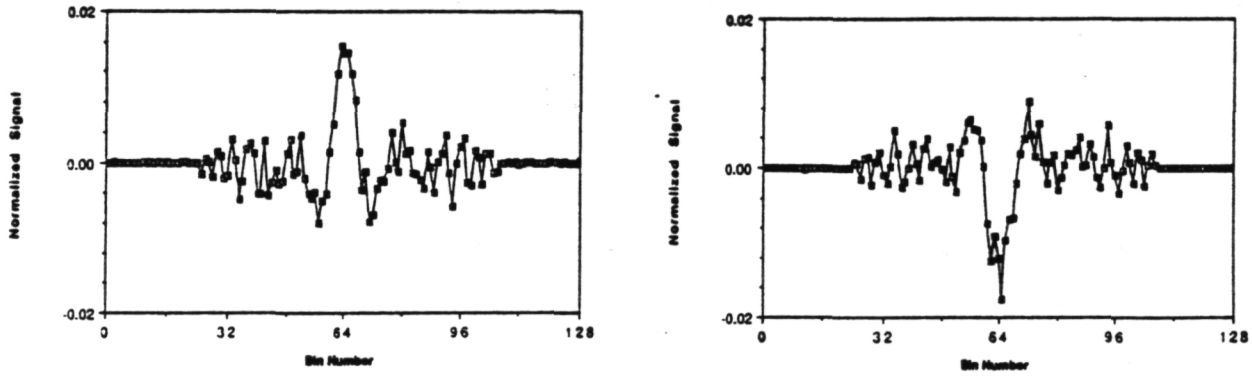
Figure 7. Calibration profile for 8000 scans.

Having set the profile intensity at an appropriate level, a series of single scans were performed to check the auto-centering function of the signal processor. These single scans were followed by tests of the normalizing and differencing functions incorporated within the control software. To perform such tests with a single source, the scan rate was first decreased, and subsequently increased, with respect to the nominal rate of 5 hertz.

* The theoretical intensity distribution associated with imaging a monochromatic point source is best described by the axisymmetric rotation of a J1 Bessel function.

When the scan rate is decreased with respect to that of a calibration scan, the apparent width of the resulting profile is increased in direct proportion to the change in scan rate. Consequently, when the slow-scan profile is normalized (so as to have the same total count as that of the reference profile) its peak amplitude is less than that of the reference profile. Thus, the difference signal exhibits a central minimum. On the other hand, at the wings of the central peak of the profile, the normalized slow scan has an amplitude which is greater than that of the normalized reference. This causes the difference signal to exhibit subsidiary maxima on either side of the central minimum. Conversely, when the scan speed is increased, rather than decreased, the difference signal exhibits a central maximum which is flanked by a pair of subsidiary minima.

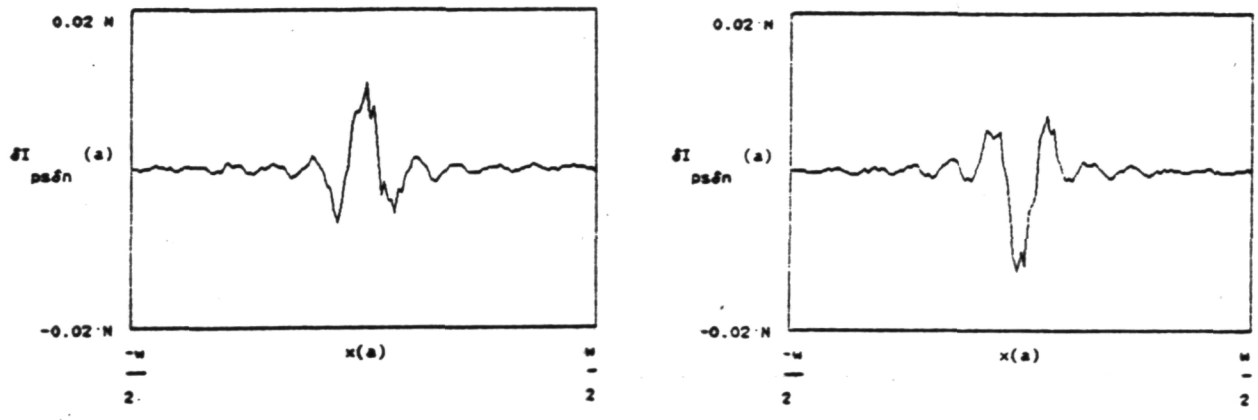
For test purposes, normalized profiles were generated by the integration of 500 scans when the scan rate errors were set at ±0.01. Subtraction of a normalized reference profile (based upon a like number of scans), produced the difference signals shown in Figure 8. It is instructive to compare these signals with those produced when identical scan rate errors and cumulative pulse counts are entered into the computer simulation program. The computer synthesized signals are shown in Figure 9. Recognizing that both the experimental data and the computer simulation incorporate random errors having a *one-sigma* value on the order of 0.002 (one-twentieth of the vertical extent of the graphs), the correlation between real and simulated data indicate that, for this level of scan integration, the breadboard profiler performs in a manner which correlates closely with that predicted by the computer model.



a. 500 fast calibration scans (+1%)/ 500 calibration scans

b. 500 slow calibration scans (-1%)/ 500 calibration scans

Figure 8. Experimental effects of scan rate errors



a. 500 fast calibration scans (+1%)/ 500 calibration scans

b. 500 slow calibration scans (-1%)/ 500 calibration scans

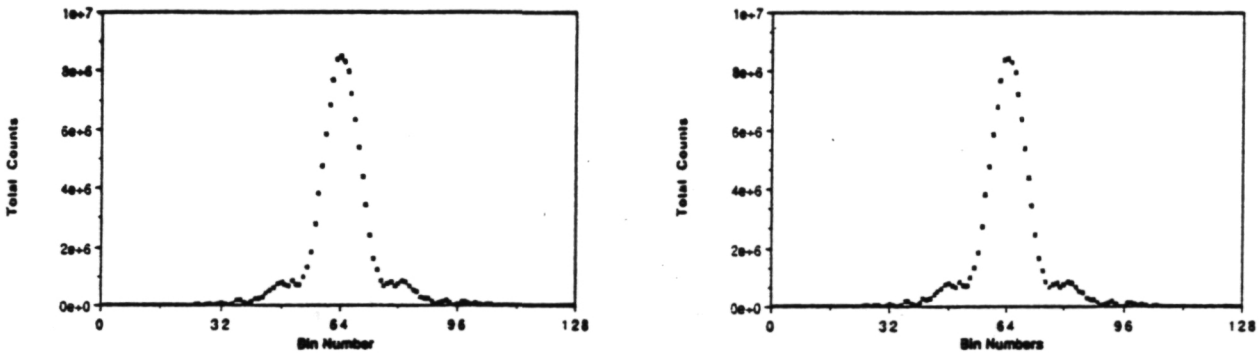
Figure 9. Computer synthesized effects of scan rate errors

Faint source detection: To simulate the presence of a faint companion source, an identical source screen was located in the second image projector. To calibrate the intensity of the companion, preliminary runs were conducted with similar attenuating filters in the two projectors. Since the companion source was introduced to the optical train via an uncoated pellicle, this arrangement created a situation wherein the intensity of the companion was approximately one-tenth that of the primary. This allowed the relative intensity of the companion to be measured with a fair degree of accuracy. For subsequent runs, wherein the intensity of the companion was greatly reduced by the introduction of additional neutral density filters, the relative intensity was computed from the filter calibration curves.

To demonstrate a breadboard performance consistent with the previously developed computer models, a series of tests were conduced with primary/secondary intensity ratios ranging from 10^{-2} to $2x10^{-4}$. For the most part, these tests were conducted with the companion source displaced from the center of the primary such that it was superimposed upon the first ring of the primary diffraction pattern. The number of scans associated with single experiments extended from a minimum of 100 to a maximum of 50,000. Within this range, it was found that the system behaved in a manner consistent with the computer model.

At a scan rate of 5 hertz, the number of scans which can be accumulated in an hour is 18,000. Thus, it was found convenient to structure experiments so as to employ a maximum of 8,000 scans both for the reference profile and for the composite profile. This involved a total scan time of slightly more than 50 minutes; allowing individual experiments to be conducted in a period of an hour, or less.

Figure 10 illustrates the nature of typical non-normalized profiles; the left-hand profile corresponding to a reference source alone; the right-hand profile corresponding to a composite profile. Here, the central bins have accumulated slightly more than 1,000 counts per scan; yielding approximately 10^8 counts within each 8,000-scan profile. The relative intensity of the companion source in the composite profile is 0.001. As expected, the presence of a companion of this magnitude is not discernible from an examination of the unprocessed data. In fact, what appears as irregularity, or noise, in the composite data, are seen to be replicated precisely in the reference data. Thus, apparent irregularities in the signal scan are attributed to the image screen, rather than to the presence either of a companion source or of system noise. In this respect, the screen provides a fair simulation of the irregularity which can be expected in the point spread function of a real telescope.



a. Total counts for 8000 calibration scans

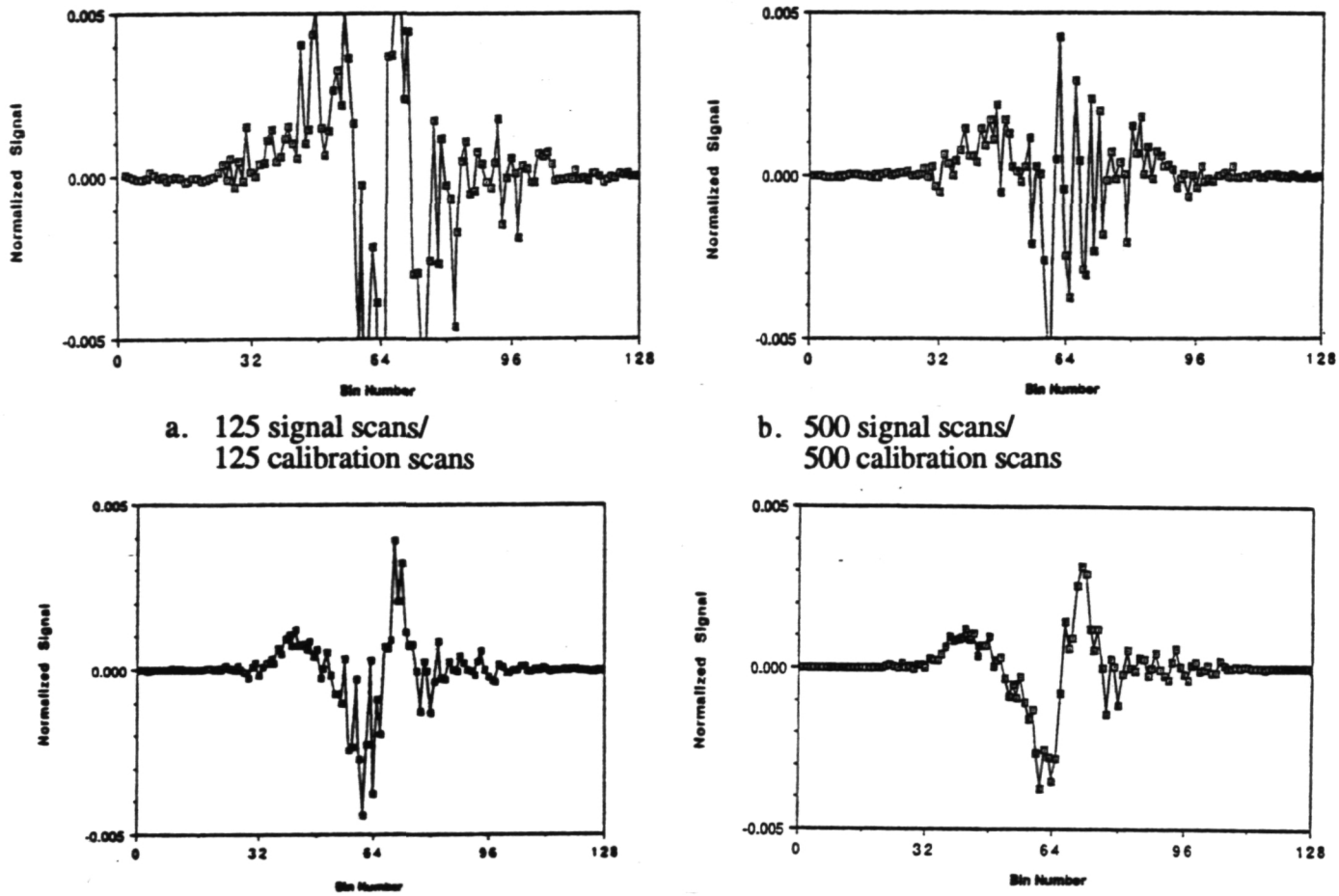
b. Total counts for 8000 signal scans

Figure 10. Comparison of integrated profiles

The *one-sigma* noise associated with an accumulated bin count of 8 x 10^6 random events is approximately 2830 counts. When subtracted from a bin count of comparable magnitude, the probable error increases by $\sqrt{2}$; yielding a *one-sigma* value for the difference of about 4,000 counts; i.e., about one part in 2000 of the individual bin count. Consequently, when a companion source having a relative intensity of 0.001 is introduced, we expect to encounter a signal-to-noise ratio in the neighborhood of 2:1. If we reduce the number of scans by a factor of four (i.e., to 2000), the signal-to-noise-ratio is expected to fall to 1:1. Likewise, reducing the number of scans to 500 and to 125 should yield signal-to-noise ratios of 1:2 and 1:4, respectively.

Figure 11 presents normalized difference signals for experimental data that illustrate the influence of scan integration time upon signal-to-noise ratio. For these data, the companion source was set at a relative intensity of 0.001 and was displaced to the left of the primary source centroid by a distance equivalent to $2\lambda/D$. The scans also included a scan error having a magnitude of 0.0015; the presence of which produces the characteristically sharp minimum at the center of the difference

profile. Recognizing the presence of a scan rate error, we see that the experimentally derived signal-to-noise ratio does, indeed, vary in proportion to the square root of the number of scans; i.e., as predicted by the theoretical model.



a. 125 signal scans/
125 calibration scans

b. 500 signal scans/
500 calibration scans

c. 2000 signal scans/
2000 calibration scans

d. 8000 signal scans/
8000 calibration scans

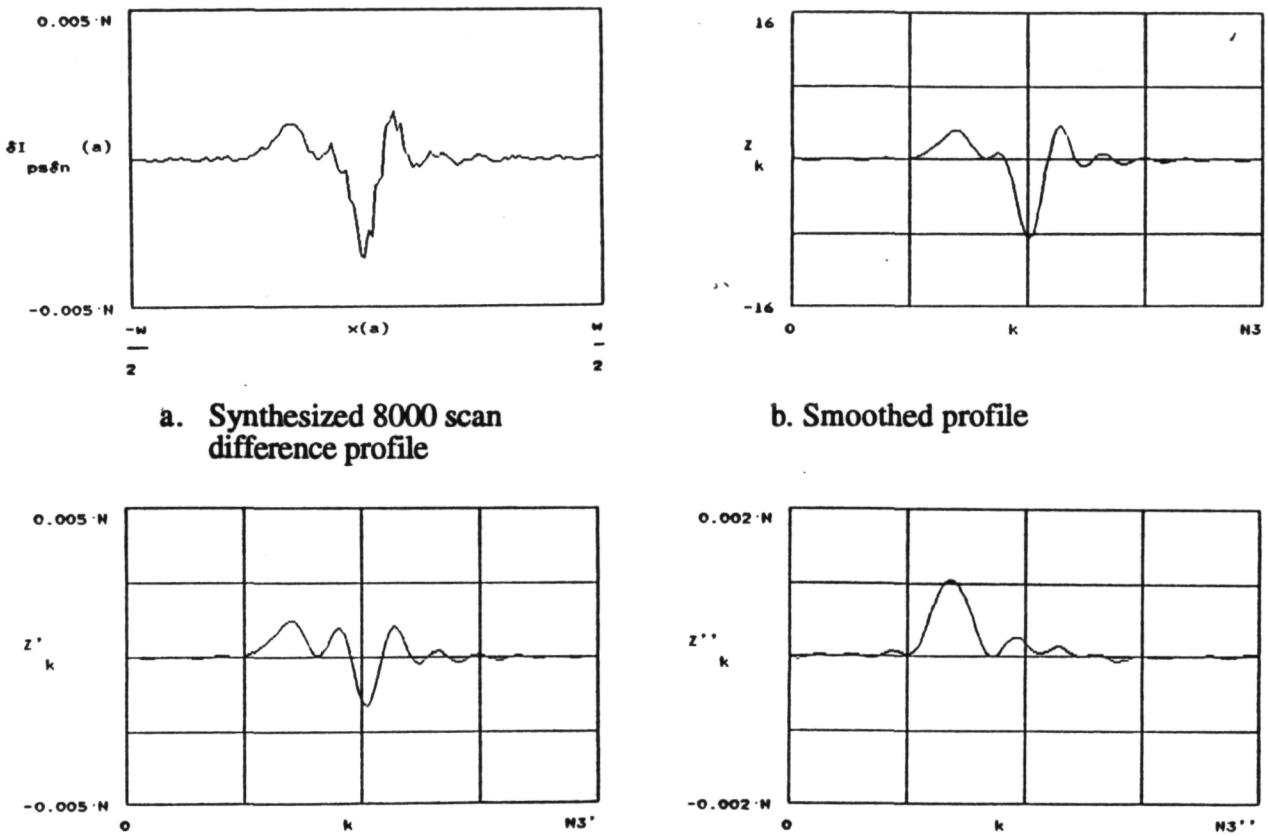
Figure 11. Influence of integration time on signal-to-noise ratio

4. Fast Fourier Transform Filtering

In the context of multichannel systems which employ fast Fourier transform filtering of difference profiles, it is important to examine the behavior of the filtering routine when faced with unprocessed data of the type shown in Figure 11. For this purpose, we employed the computer synthesized profile shown in Figure 12a. Use of a synthetic profile permits a more precise knowledge of the input parameters which define relative intensity and angular position of the companion source. This, in turn, provides a better assessment of the accuracy with which the data processing system determines such parameters. The specific parameters used to generate the synthetic profile were tailored to the conditions of the 8000-scan data of Figure 11 and are as listed in Table III.

Table III. Synthetic profile parameters

Scan width	12λ/D
Number of scans	8,000
Scan rate error	-0.0015
Number of bins	128
Maximum counts/bin/scan	1,000
Relative intensity of companion	0.001
Displacement of companion	-2λ/D



a. Synthesized 8000 scan difference profile

b. Smoothed profile

c. Recentered profile

d. Recentered and scan error corrected profile

Figure 12. Fast Fourier filtering and correction of a signal profile

The multistep fast Fourier transform filtering process was implemented in MathCAD and is completely deterministic. The first step of the filtering process involves the removal of high-frequency noise. This is accomplished by taking a fast Fourier transform of the raw profile, removing high-frequency terms from the transform, and inverse transforming the result. This step produces the smoothed profile shown in Figure 12b.

The second step of the process performs a first-order refinement of the reference profile by correcting for the apparent amplitude and location of the companion source. Having corrected the reference profile in this manner, a new difference profile is created. The new profile then is refiltered by the fast Fourier process to produce the profile shown in Figure 12c. The profile of Figure 12c exhibits the presence of a companion source, together with a scan rate error, the latter being evidenced by a characteristic central minimum which is flanked by subsidiary maxima. The scan error is corrected automatically by a second refinement of the reference profile. The twice-refined reference profile is then used to produce the Fourier transform filtered difference signal shown in Figure 12d. (Note that the vertical scale of Fig. 12d has been expanded by a factor of 2.5 relative to the scale of Figs. 12a, 12b and 12c.)

Finally, the profile of Figure 12d is computer analyzed to establish best estimates for the relative intensity and the angular position of the companion source. For the specific parameters used in this example, the relative intensity and position were estimated by the processor to be 0.00106 and $-1.92\lambda/D$, respectively; i.e., to accuracies of 6 percent and 4 percent, respectively.

As noted previously, the signal-to-noise ratio expectation for an unfiltered image profile, taken under the conditions assumed above, is on the order of 2:1. However, it is evident that a filtering process which takes advantage of interbin correlation enhances this ratio by a significant factor. Thus, in theory, the marriage of a multichannel profiler and a fast Fourier processing system provides a level of performance which exceeds that which was predicted prior to embarking upon the Phase I program.

III. DESIGN AND FABRICATION OF THE MULTICHANNEL SYSTEM

To satisfy the need both for a laboratory brassboard and for a prototype instrument suited for use in the field, it was necessary to design and construct a self-contained and portable instrument. To achieve this objective during Phase II, we adopted a design approach wherein the resulting system comprises three basic packages. The first package, which is referred to as the head unit, contains the entire optical subsystem, together with the photomultiplier tube array. The second package, referred to as the electronics unit, contains all power supplies plus the hardwired portion of the signal processing system. The third package, referred to as the control unit, consists of a computer which is equipped with specialized I/O boards. By dividing the packages such that their interconnection is made either via conventional coaxial cable and/or via low bandwidth ribbon conductors, the separation of the electronics and the control unit from the head unit is made large enough to accommodate the working environment of a typical astronomical observatory.

All control functions are controlled via the computer. Mechanical functions of the head unit which are controlled in this manner include image focus, image magnification, image rotation, optical attenuation, and scan speed. Additionally, all functions related to scan integration time, correction for seeing conditions, signal profile filtering, etc., are computer selectable.

The head unit is designed so as to be compatible in all respects with NASA's 24-inch telescope on Table Mountain. Since the focal plane mounting interfaces of all telescopes on Table Mountain are identical, this approach also makes the profiler compatible without modification to the larger 41-inch and 48-inch telescopes. However, detailed design of the head unit was conducted primarily with the characteristics of the 24-inch telescope in mind.

As shown during Phase I, the dynamic range of a profiler can be maximized partly by the use of input prescalers and partly by using a multichannel collective image centration logic. Here, the term *dynamic range* is used to express the acceptable intensity range for the primary source, and is unrelated to the ratio of primary to secondary intensity. The use of divide-by-ten prescalers raises the upper limit of the dynamic range by about an order of magnitude. For bright primary sources, this provides a corresponding reduction in the time required to achieve a specified threshold of sensitivity for faint companion detection. By allowing the collective counts within a group of eight channels to establish an image centroid, the counts per channel required to achieve a specified centration accuracy is reduced by a factor of eight. Recognizing that a single-channel system without a prescaler provides a dynamic range of about 10:1, the expanded dynamic range of the multichannel system is about 1000:1; i.e., encompasses about seven stellar magnitudes. This allows the 24-inch

Table Mountain telescope to operate effectively with primary sources having stellar magnitudes within the approximate range extending from $m_v = 1$ to $m_v = 8$.

As noted previously, the use of a prescaler also allows the shift register clock rate and the capacity of individual shift registers to be reduced dramatically relative to those used in the Phase I breadboard. Thus, in place of the 20-megahertz clock and 1-megabit register capacity of the breadboard, each channel of the multichannel brassboard incorporates the equivalent of a 64-kilobit register operating at a clock rate of 640 kilohertz.

For image scanning purposes, the head unit employs a multifaceted aluminum mirror mounted upon a microstepping motor. Using a 50-facet mirror with a driver having 25,000 microsteps per revolution provides approximately 500 steps within the angle of a single scan. The scanning speed of the brassboard is variable; permitting the use of discrete frame rates of 8, 16,and 25 hertz. Since the duty cycle of individual channels falls in the vicinity of 0.5, these rates are equivalent to single-channel scan times of 62.5, 31.25, and 20 milliseconds, respectively. This variable scan rate feature is important in the context of operation on a ground based telescope, since the use of shorter scan times are required to freeze the effects of atmospheric distortion.

To take advantage of shorter scan times, the control software can incorporate a simple profile assessment routine whereby the cumulative profile can be optimized with respect to seeing conditions. When implemented, this routine periodically determines the fraction of individual scans for which the ratio of central bin content to total scan content exceeds a set of predetermined ratios. Essentially, this is equivalent to establishing a probability distribution for the finesse, or one-dimensional Strehl intensity ratio of the atmosphere. Having established such a distribution, the operator can elect to accumulate only those scans for which the finesse exceeds a specified value.

To describe in greater detail the activities associated with the design and fabrication of the Phase II multichannel brassboard system, it is convenient to divide the ensuing material among three topics. These topics are entitled Optomechanical design, Signal processor electronics design, and Software development.

1. Optomechanical Design

An optical layout of the multichannel brassboard differs from that of the single-channel breadboard in three respects. First, it is designed to profile an optical image at the focal plane of a real telescope. Second, it uses a scanning mirror system in place of a multi-lens scanner. Third, the single-channel fiber optic assembly is replaced by a multichannel assembly. The overall layout of the re-

sulting optomechanical subsystem is shown in block diagram form in Figure 13. The completed subsystem, when mounted upon its test stand, is presented in Figure 14. To maximize accessibility to both the front end optics and the photomultiplier tube array, the subsystem is designed to be opened via the simple removal of two clamping rings. Figure 15 shows the instrument in its opened state.

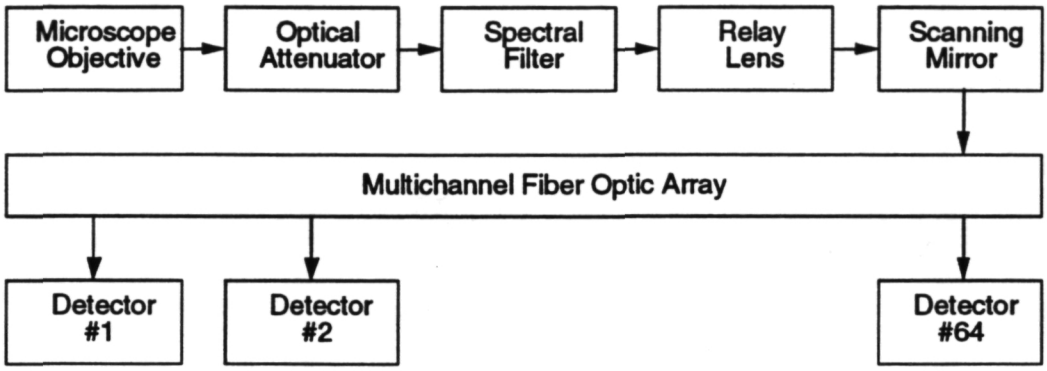


Figure 13. Optomechanical subsystem block diagram

A prime consideration in the evolution of the optical layout was that of achieving compatibility with the 24-inch telescope on Table Mountain. In particular, close attention was paid to matching the field of view of the profiler to the focal lengths afforded by the Cassegrain and Coudé foci of the telescope. The f-ratios at these foci are 16 and 36; yielding focal lengths of 975 centimeters and 2195 centimeters, respectively.

Since the fiber optic array at the focal plane of the profiler employs the equivalent of 64 contiguous slits which are 0.25 millimeters wide, the scan width at the array is 16 millimeters. Since image scales both greater and smaller than the nominal scale can be provided by interchangeable objectives, a nominal scale on the order of 3 microradians per millimeter at the array is considered convenient. This provides a nominal field of view of 48 microradians; i.e., approximately 10 arc seconds. The effective focal length required to achieve such an image scale is 3.33×10^4 centimeters. Thus, to provide the nominal field at the Cassegrain focus requires that the internal magnification of the profiler be 34.2X. Likewise, to provide the same field at the Coudé focus, the internal magnification must be 15X.

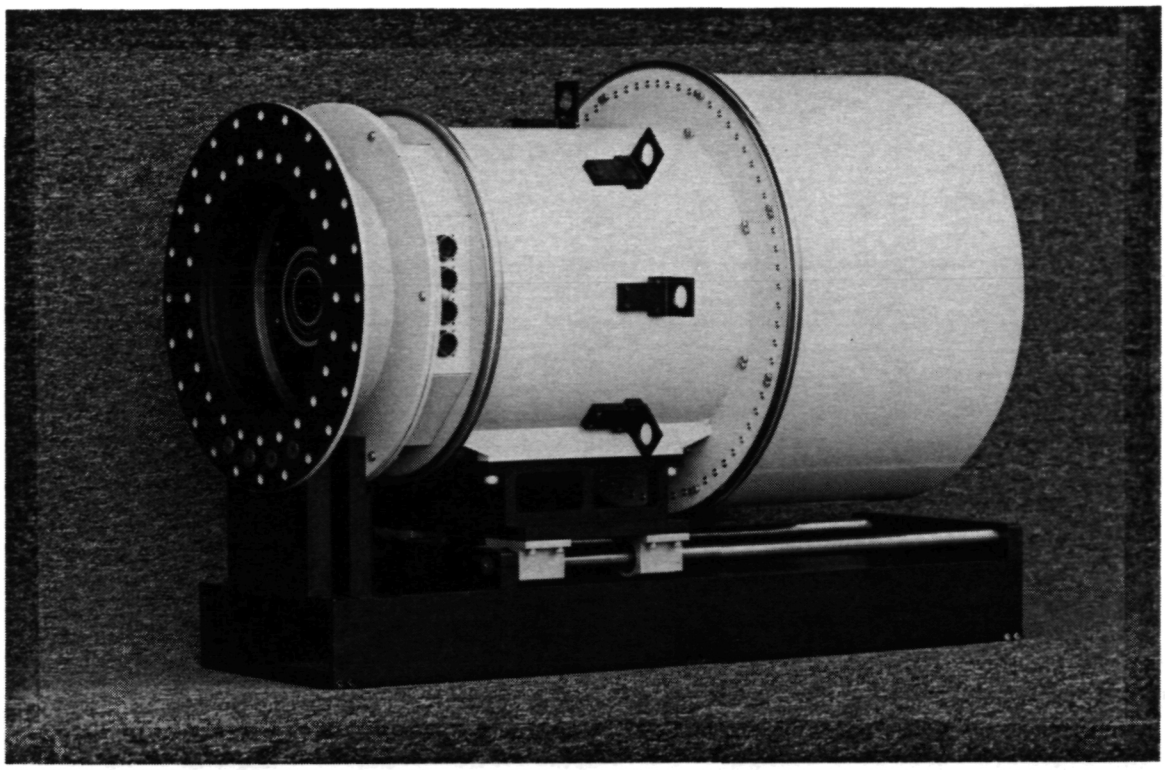


Figure 14. Optomechanical subsystem mounted on test stand

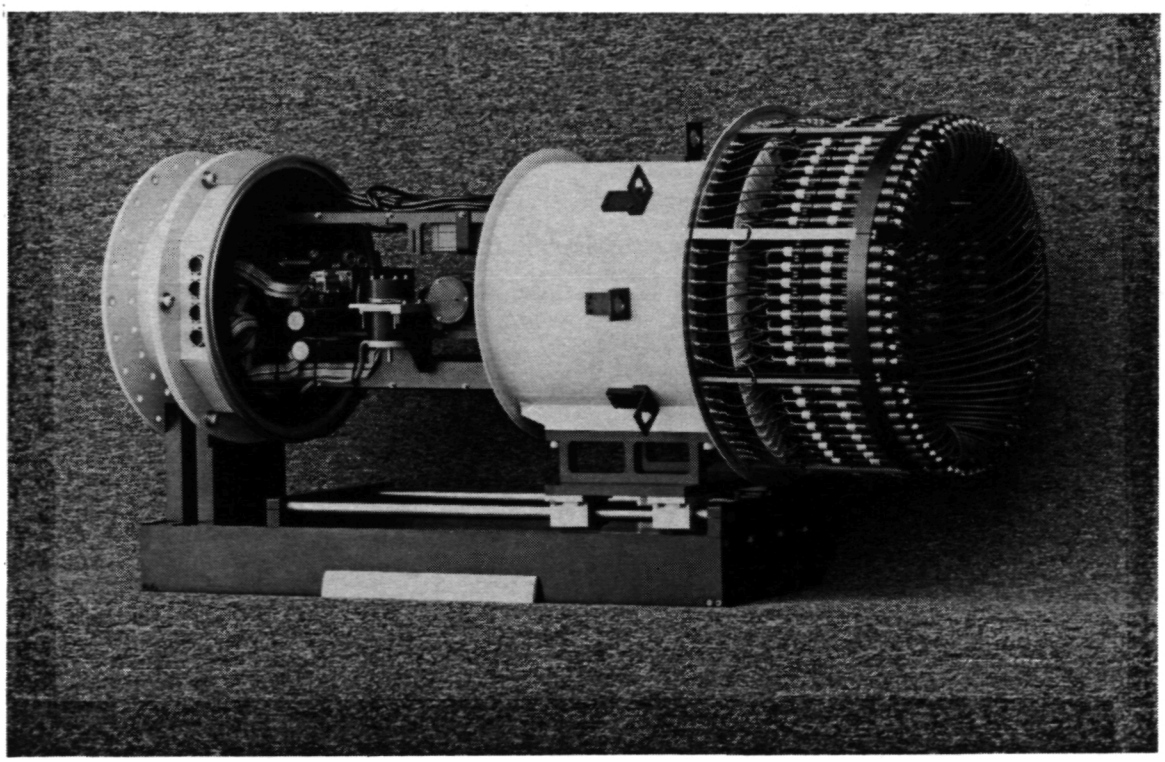


Figure 15. Optomechanical subsystem opened for servicing

By using parfocal microscope objectives at the input to the profiler, the internal magnification can be increased in steps of two to one. The selected set of four objectives, designed to operate with a back focal distance of 160 millimeters, exhibit nominal focal lengths of 8 millimeters, 16 millimeters, 32 millimeters, and 64 millimeters. By placing an appropriate relay lens between the back focal plane of the microscope objective and the final focal plane of the profiler, the 32-millimeter objective provides an internal magnification of 16X; thereby approximating the desired nominal field at the Coudé focus. Meanwhile, the 16-millimeter objective provides an internal magnification of 32X and approximates the desired field at the Cassegrain focus. Using the set of four objectives listed above on a precision turret provides the combination of fields listed in Table IV below.

Table IV. Profiler fields of view relative to the Table Mountain 24-inch telescope

Profiler Objective Focal Length (mm)	**Telescope Field of View (microrad/sec)**	
	Cassegrain Focus	**Coudé Focus**
8	25.6/5.2	11.3/2.6
16	51.2/10.5	22.7/4.7
32	103/21	45.5/9.4
64	205/42	91/18.8

To achieve the above fields of view, the magnification of the relay lens within the profiler must be 3.2X. This relay function is performed via a pair of achromatic lenses; one located ahead of the scanning mirror; the other beyond the mirror. Axial placement of the first lens is such that the entrance pupil of the telescope is imaged upon the surface of the scanning mirror. This arrangement ensures that, despite the use of a large magnification, the pupil location at the scanning mirror is not influenced by tracking errors. Also, for typical magnifications, it results in a scanning mirror pupil which is less than 0.5 millimeter in diameter, i.e. is consistent with efficient scanning for mirror facets which are approximately 5 millimeters in tangential extent. The second lens, having a focal length which is approximately 3.2 times that of the first lens, is located such that its forward focal plane coincides with the surface of the scanning mirror, while its rear focal plane coincides with the input face of the fiber optic array. This causes the chief ray of the incoming light bundle to fall at normal incidence to the fiber optic array while, at the same time, preserving linearity of the scan geometry.

The brassboard fiber optic array employs entrance slit widths of 0.25 millimeter. Construction is such that each fiber bundle has its rectangular end potted in the form of a ribbon; the thickness and width of which are 0.25 millimeter and 25 millimeters, respectively. Each bundle contains approximately 5000 fibers; the relative positions of which are randomized prior to being molded to a circular cross section at the opposing end. Precision potting at the rectangular end enables 64 bun-

dles to be stacked and clamped so as to form a flat entrance plane with overall dimensions of 16 millimeters by 25 millimeters. Consistent with the 8 x 8 modularity of the overall system, actual construction involves discrete groups of eight fiber optic channels; each group being potted as a separate subassembly. The mechanical layout of this modularized structure is shown in Figure 16.

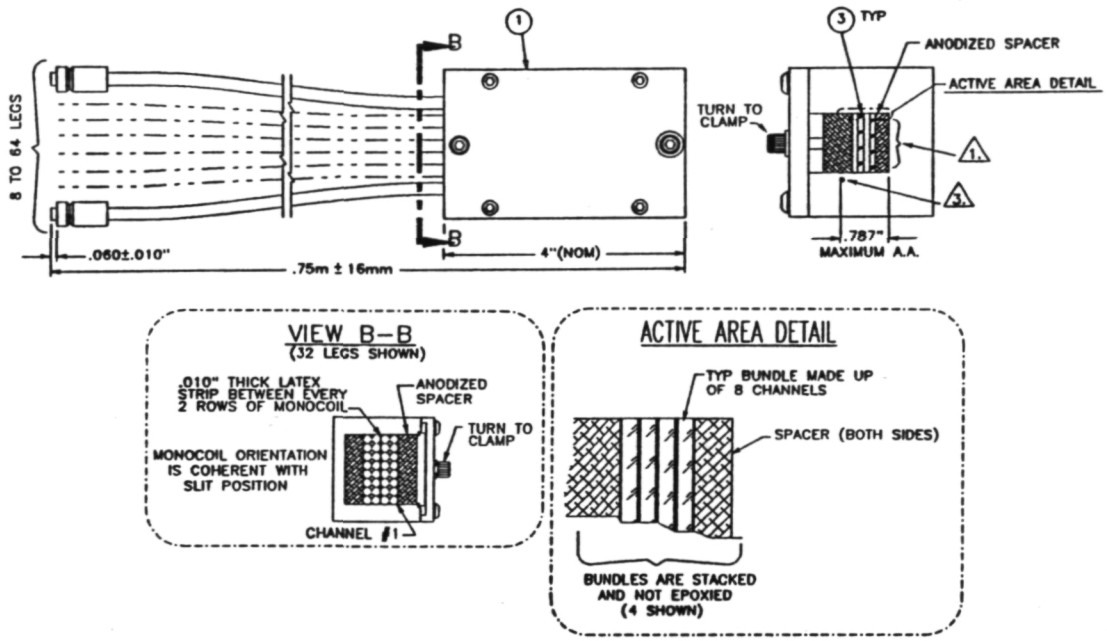


Figure 16. Mechanical layout of the modularized 64-channel fiber optic assembly

Prior to conducting a detailed design of the brassboard, it was envisioned that remote control of system magnification, optical attenuation, and image rotation would be achieved via separate rotary positioning devices. However, when combined with auto-tracking and auto-focus, such an arrangement requires a total of six remotely controllable devices. On the other hand, our cost effectiveness study showed that the optimum number of positioning control channels is four. Reduction of control channels from six to four was achieved by the adoption of redundant attenuator/microscope objective sets and by using a fine adjust motion of the microscope objective as a means for providing auto-track in one axis. Consistent with this design philosophy, we adopted a 12-position microscope objective turret which is provided with a vernier positioning capability.

The 12-position objective turret permits four magnification settings to be combined with three optical attenuator settings. For example, three separate 10X objectives are used; one without a filter, one in combination with an ND-1 filter and one in combination with an ND-2 filter. To accomplish image tracking in one axis, the geometry of the filter/microscope objective turret is arranged such that rotation of the turret causes the objective to translate along a curved path which lies at right angles to the incoming light beam. For small angular rotations, the resulting tangential motion

closely approximates that required for one axis of the tracking function. Implementation of the four axis assembly which simultaneously controls magnification, attenuation, image rotation, image tracking and focus is illustrated in Figure 17.

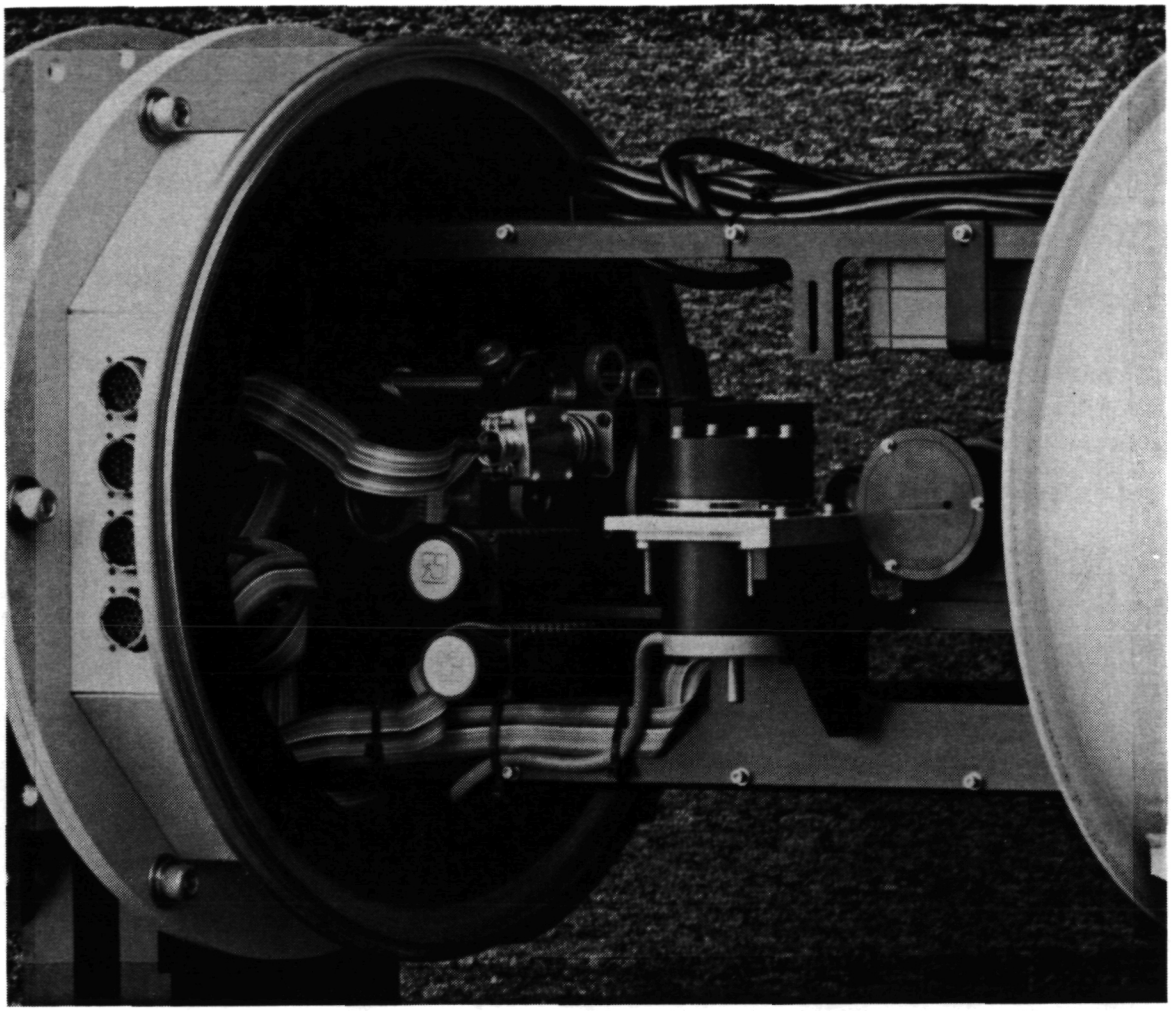


Figure 17. Close up of the four-axis assembly which controls magnification, attenuation, image rotation, image tracking, and focus

The breadboard profiler employed a 50-millimeter photomultiplier tube. However, the brassboard employs an array of smaller Hamamatsu type R647-04 tubes. These tubes are specifically designed for photon counting applications and have an overall diameter of 15 millimeters. This diameter includes magnetic shielding and exterior conformal coating. Such a compact design permits 64 tubes to be arranged in a one-deep circular array having a diameter of approximately 40 centimeters. Preserving the overall 8 x 8 module philosophy, the final design incorporates a set of eight 45-degree detector subassemblies (see Figure 18).

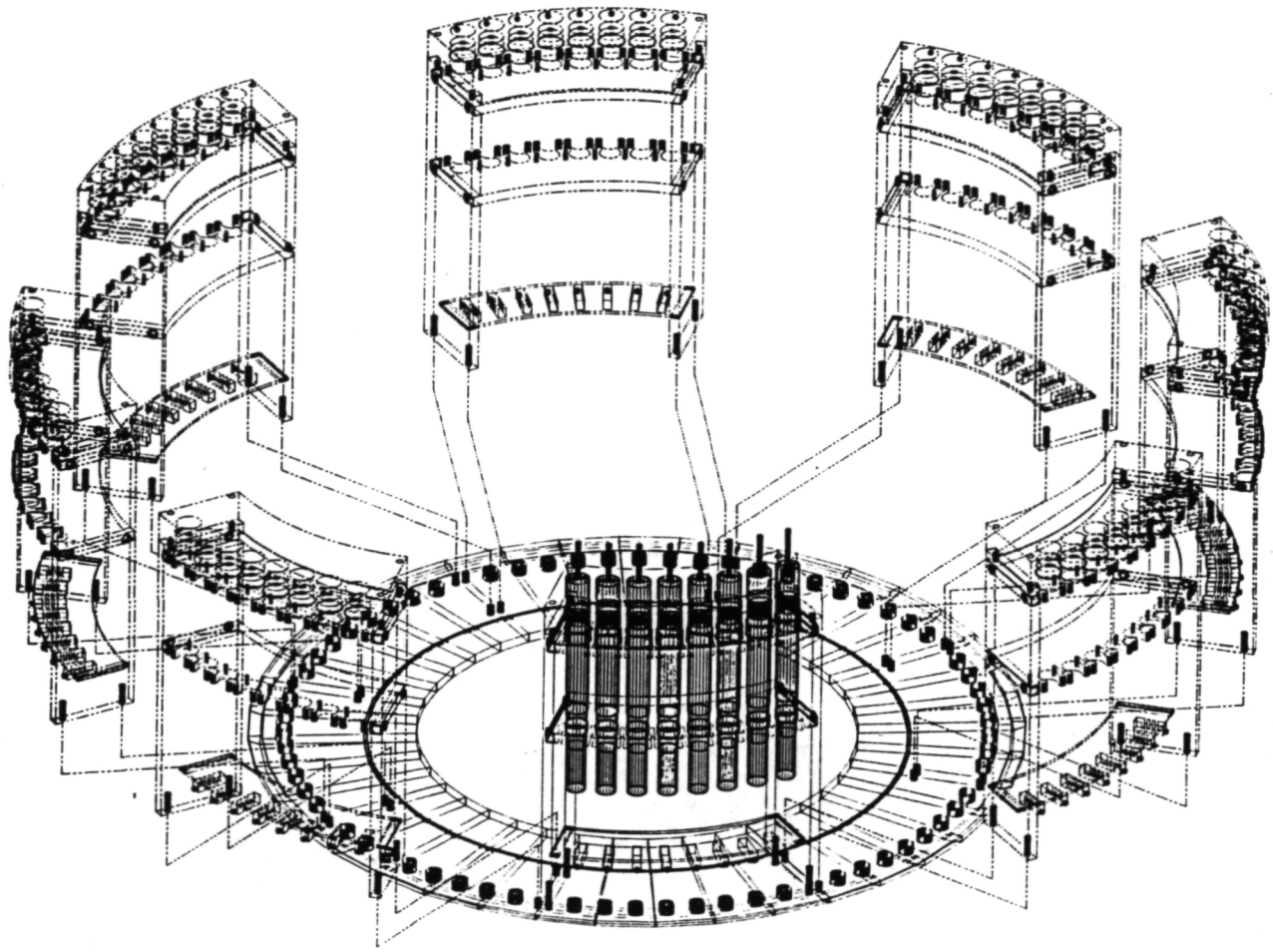


Figure 18. 8 x 8 modular detector geometry

To aid both in target acquisition and in general monitoring of optical performance, a CCD camera was added to the original optical layout. This CCD initially was located at the focal plane which lies behind the microscope objective. However, at this location the field of view of the CCD is extremely small; typically on the order of 10-20 arc seconds, depending upon the selected magnification. Early field tests showed this narrow field to be impractical for the purpose of target acquisition. Consequently, we modified the system so as to place the CCD at the true Cassegrain focus of

the telescope; i.e. ahead of the microscope objective. This was achieved by mounting a 45-degree mirror on the magnification selecting turret. The geometry of this modification is shown in Figure 19.

In operation, the diagonal mirror is rotated into the optical path for image acquisition purposes only. Calibration within the laboratory established three-axis coordinate differentials between the focused camera condition and the appropriate locations of each microscope objective/filter combination. Storage of position differentials in software permits interchange between different magnifications to be performed without loss of image centration or focus.

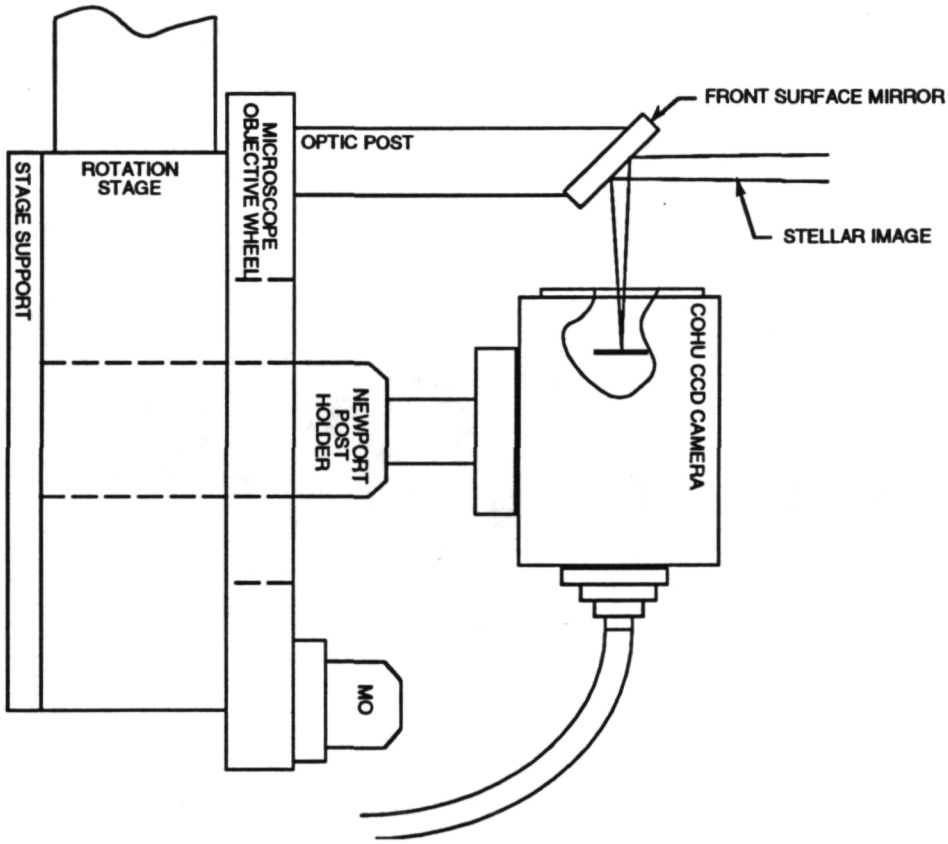


Figure 19. Optical train modification for inclusion of a CCD ahead of the microscope objective

The completed fiber optic/photomultiplier assembly is shown in Figure 20. Here, the 64 individual fiber optic cables are shown plugged into the right hand side of the photomultiplier tube array. The photomultiplier tubes, together with their sockets and potted dynode resistor chains, are separately removable from their plastic mounting clamps. To the left in Figure 19 are the multi-turn high voltage adjustment potentiometers and the 64 individual coaxial signal connectors. Use of a separate potentiometer for each photomultiplier tube permits the gain of all tubes to be matched during system calibration. A separate coaxial signal line from each channel is necessary for preservation of

the composite signal bandwidth. Since each channel exhibits a 150MHz bandwidth, the composite system bandwidth approaches 10GHz.

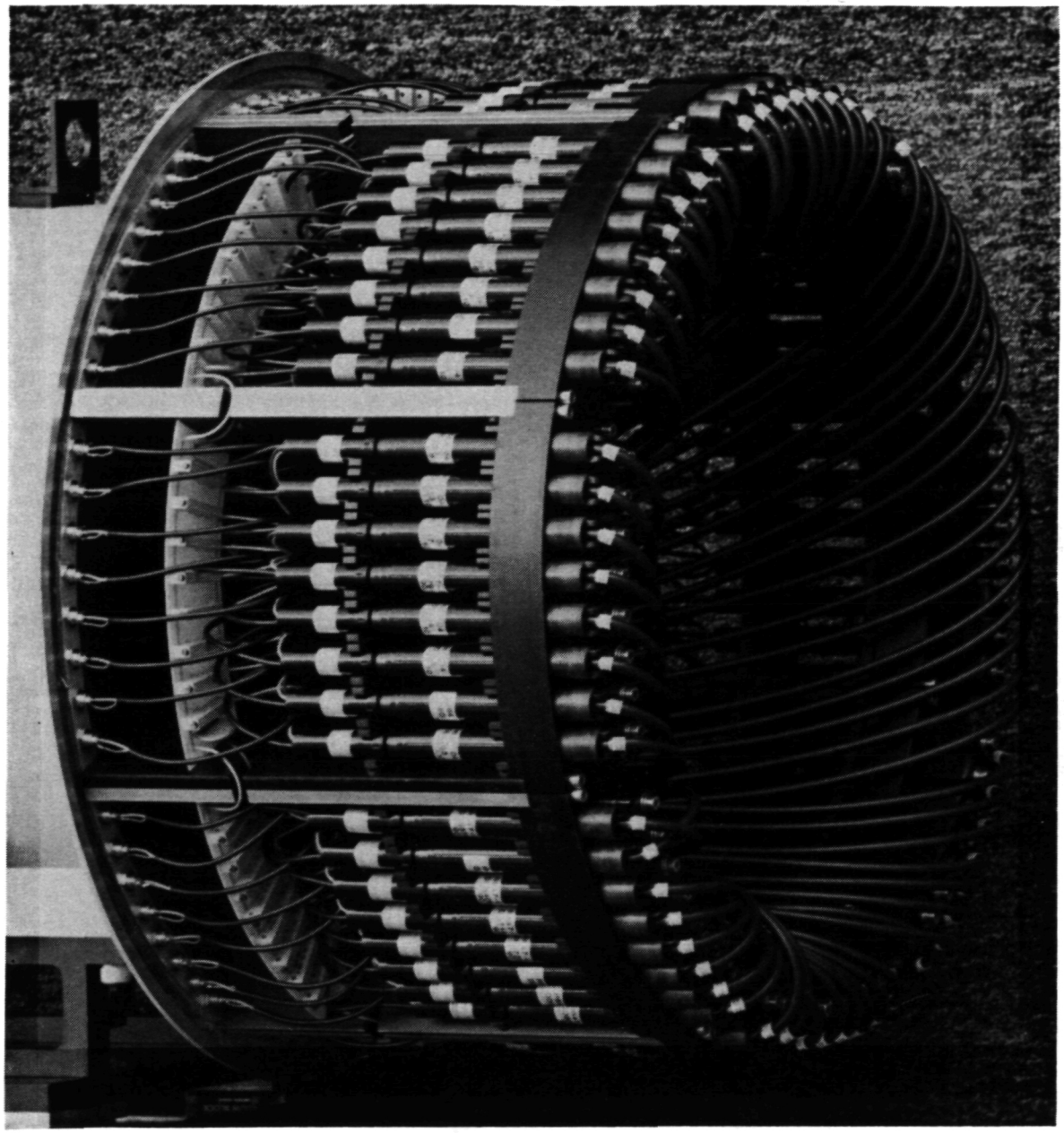


Figure 20. Fiber optic/photomultiplier array with cover removed

2. Signal Processor Electronics Design

A general overview of the signal processor electronics is presented in Figure 21. As an image is scanned across the linear array of optical fibers, a modulated signal is transmitted to the face of each photomultiplier tube. The photomultiplier tubes (PMTs) generate narrow analog pulses; the

mean rate of pulses being proportional to the detected light intensity. The PMT pulses are amplified by high speed CAMAC* (Computer Automated Measurement and Control) compatible analog amplifiers. The amplified output signal is input to CAMAC compatible high speed discriminators. These discriminators feature a programmable detection threshold. If the instantaneous analog input signal level exceeds a preset threshold, the ECL level digital output signal is asserted for approximately 5 nS; i.e., the detected signal is modified to a square wave ECL level output suitable for input to the signal averaging electronics.

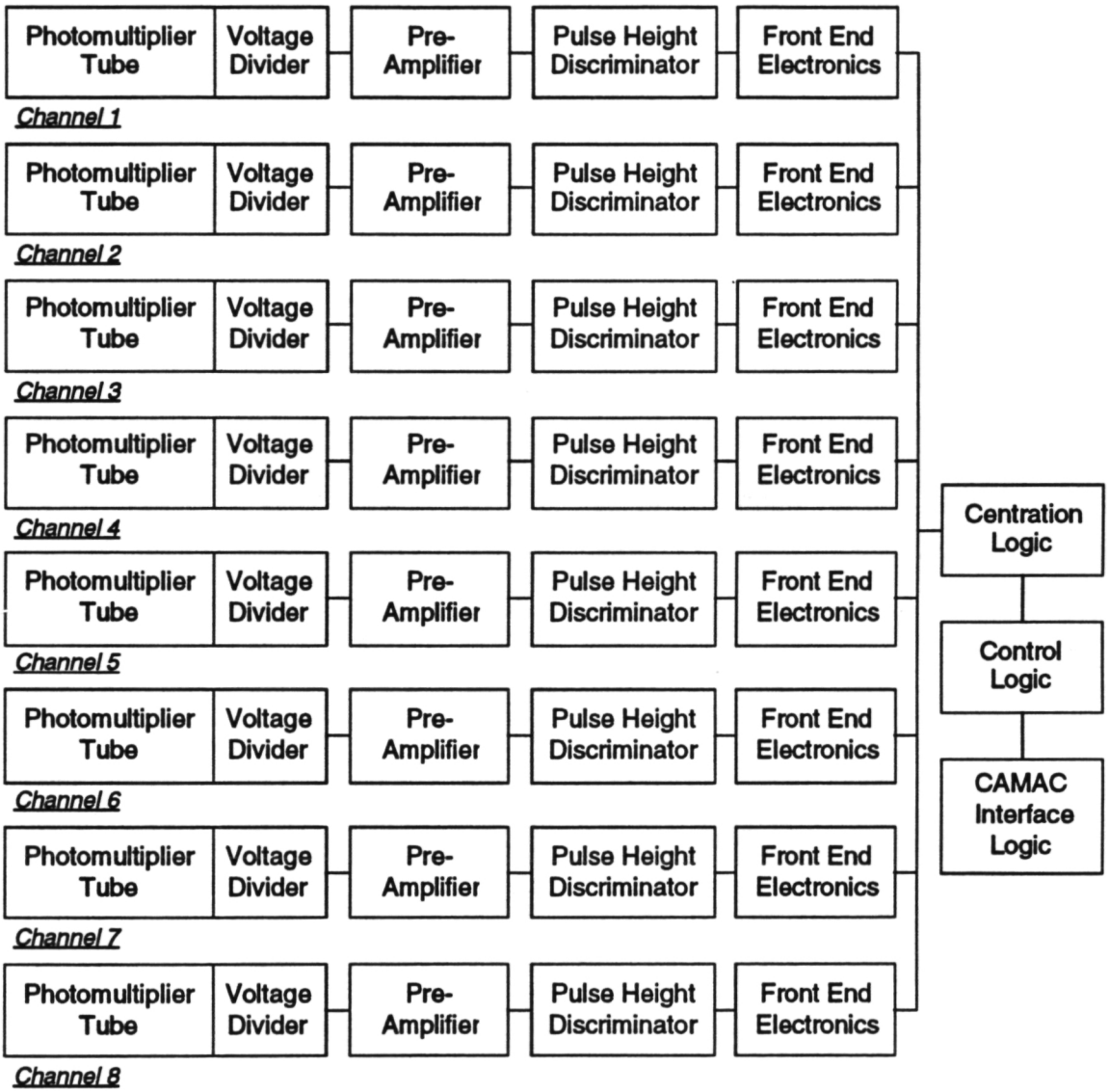


Figure 21. General overview of signal processing electronics

* CAMAC is a standardized modular instrumentation and digital hardware standard used chiefly for Nuclear Physics data acquisition applications. The CAMAC standard determines electrical and physical specifications for the modules as well as timing and protocol specifications. CAMAC hardware was selected for the data acquisition instrumentation due to the abundance of wide bandwidth electronics available in CAMAC and for the modularity and computer independent nature of the standard.

Pulses detected by the discriminators are input to the signal processing electronics developed during the Phase II program. The signals are loaded into a shift register which has counters to measure the number of individual pulses received. These electronics perform a median calculation to determine the center of the waveform. When the pulses are centered within the shift register, the samples are read out in groups. For proper image centering to occur, the pulse counters clear the shift register during readout and clear the pulse counters before acquiring the next image. The number of pulses in each group represents the intensity of the light detected by the PMT during the scanned interval. The array of data is a representation of the one-dimensional scanned image.

Since every PMT sees the same image at a different point in time, a variable delay on each channel corrects for the temporal offset. This delay also accounts for slight variations in optical fiber bundle thickness. The front end electronics, including the programmable delay, are shown in block diagram form in Figure 22.

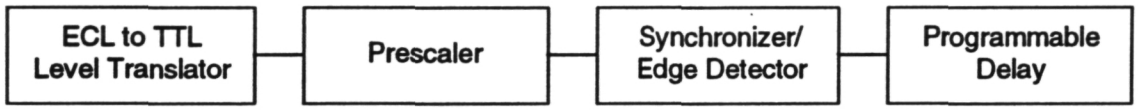


Figure 22. Front end electronics block diagram

Each digital electronics circuit board contains the processing circuits for eight channels. The input signal to the digital electronics are ECL pulses. To convert these pulses to TTL levels, a signal level translating front end was implemented. The signal averaging electronics employ the CAMAC standard. The block diagram of a complete 8-channel circuit board is shown in Figure 23. The following subsections discuss the functionality of the preamplifiers, pulse height discriminators, level translators, prescalers, synchronizer/edge detectors, programmable delays, shift register, pulse counter/center finder, control logic, clock signal generation, bits per bin counter, bin counter, state machine control, FIFO output, CAMAC interface and power supply.

Preamplifier: Output signals from each photomultiplier tube are amplified by Type 7177 CAMAC standard variable gain preamplifiers which are produced by Phillips Scientific. Since each plug-in preamplifier module contains 8 individual preamplifiers channels, the arrangement remains consistent with the 8-module philosophy adopted for the entire system. Specifications for the preamplifiers are presented in Table V.

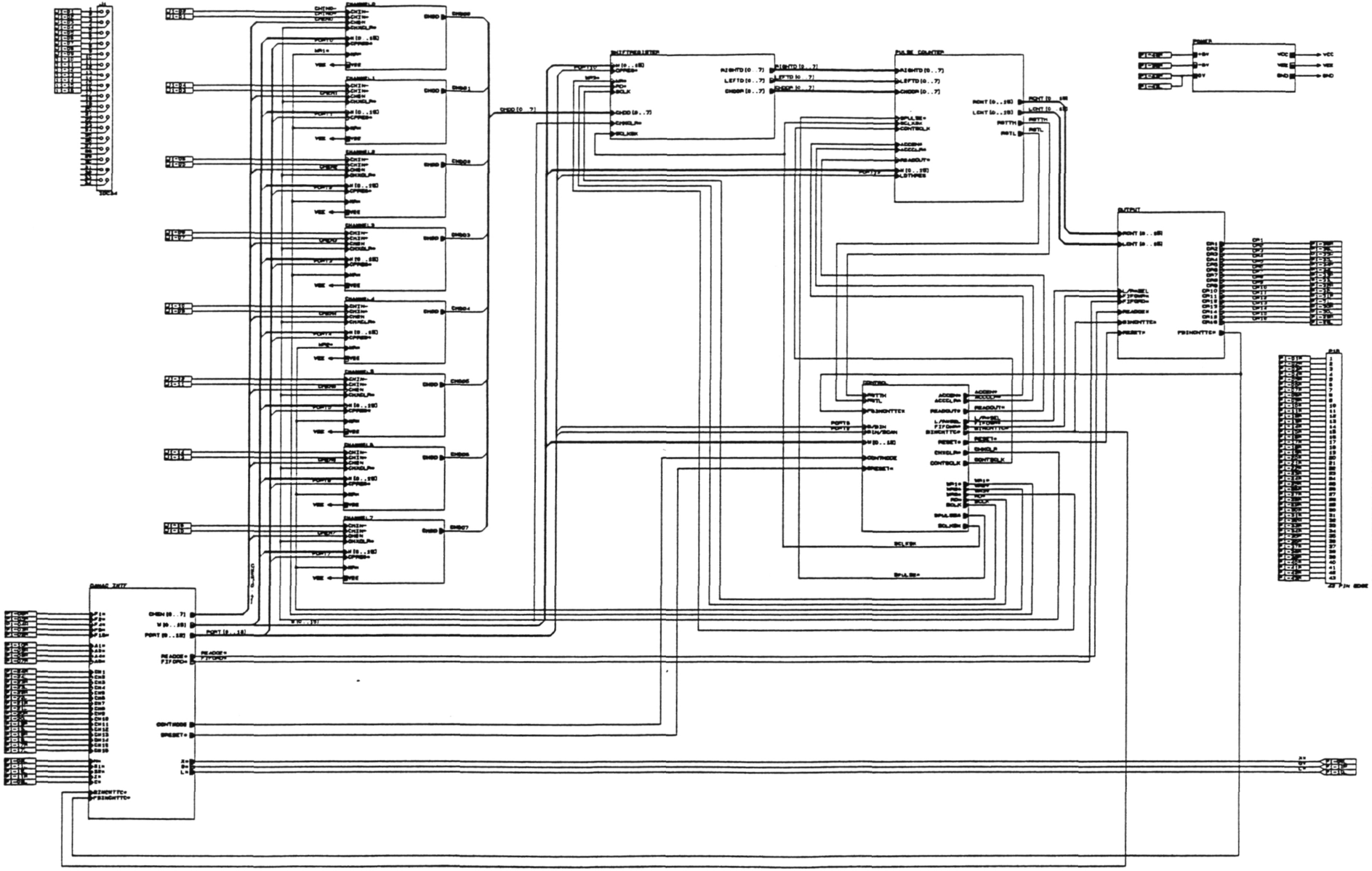


Figure 23. Block diagram of 8-channel circuit board

THIS PAGE INTENTIONALLY LEFT BLANK

Table V. Variable gain preamplifier specifications

Bandwidth	2 to 50 MHz, 3 dB point
Gain	2 to 50
Stability	±10 μVolts/°C
Channels	8
Linearity	±0.15% to -3 Volts, DC to 100 MHz
Risetime	<1.8 nSec
Crosstalk	>60 dB, DC to 100 MHz
Input/output delay	4.5 nsec Typical
Offset Control	±10 mVolt

Pulse height discriminators: Subsequent to preamplification, the signals associated with individual channels are further processed via pulse height discriminators. Again, choice of CAMAC compatible Phillips Scientific Type 7106 discriminator latch modules (each containing 16 separate channels) preserves the overall system modularity. Specifications pertaining to the pulse height discriminators are presented in Table VI.

Table VI. Discriminator specifications

Bandwidth	125 MHz
Threshold	-10 mV to -1.033 Volts
Channels	16
Signal output	Two ECL outputs per channel
Input hysteresis	4 mV
Operation modes	Local or remote

Level translator: The differential ECL pulses from each discriminator are received by the MC10H125 ECL to TTL level translators, which represent the inputs to each of the in-house developed digital circuit boards (see Figure 24). A 100 Ohm resistor provides an approximately matched load termination for the signal. The TTL level signal is passed through a resistor network to shift the voltage swing to a level acceptable to the ECL prescaler when powered by +5 volts.

Prescaler: Pulses received are as narrow as 5 nS at a rate up to 1 mega-pulses per second average. A prescaler is used to reduce the rate of the pulses and to make a wider signal that is suitable for further processing. The MC12013 is an ECL prescaler hardwired to divide by 10 (see Figure 24). This device is powered by +5 volts, so the clock input level and ECL output signal levels are not conventional ECL signal levels. The differential output (Q and Q*) drive a level translator contained within the chip to result in a TTL level signal. Divide select lines E1 to E2 are tied high to force the divide by 10 mode. For a periodic wave, the prescaler output would be a constant frequency square wave at 1/10th the input frequency. Because the PMT pulses are randomly spaced, the output of the prescaler is a waveform that changes state every five input pulses, but is not periodic.

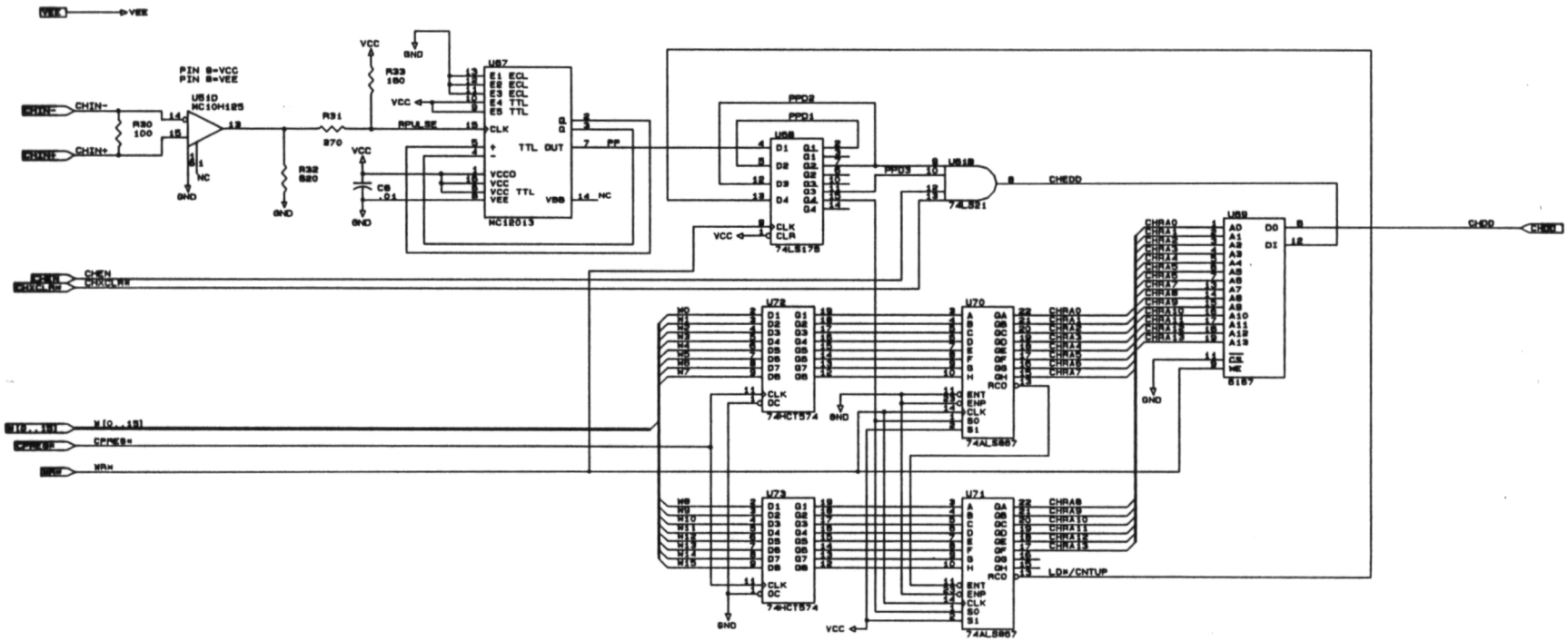


Figure 24. Level translator, prescaler, synchronizer/edge detector and programmable delay

Synchronizer/edge detector: Randomly spaced PMT pulses arrive asynchronous to the system clock. Three D type flip/flops and an AND gate form a synchronizer and leading edge detector (see Figure 24). The first flip/flop reclocks the prescaled signal with the digital electronics sample clock. At times, this flip/flop goes into a metastable state because the D input is not synchronous with the clock input. Occasionally the timing requirements of the device are violated, causing the output to go into an indeterminate state for a period of time. (No damage is done to the device as a result of a timing violation.) This state is resolved to either a high or a low after a period of time which is greater than the normal propagation time of the flip/flop (on the order of 100 nS). The period of the sample clock is much longer than this interval. Consequently, the output of the first stage is resolved by the next clock edge when it is sampled by the next stage. Thus, the second flip/flop always has a valid and stable data input when the clock occurs. Consequently, its timing is never violated. The third flip/flop delays the signal by one more sample clock.

The AND gate uses the output of the second and third flip/flop to generate a one clock wide pulse on the leading edge transition of the prescaler output. Two additional inputs to the AND gate are used to selectively disable each channel and to disable all of them together. An individual channel is disabled if a problem occurs with the circuitry leading up to the digital electronics or the PMT. All channels are disabled during memory clearing operations.

Programmable delay: The programmable delay allows all eight channels on a board to be deskewed to compensate for the shift in time as the image is scanned across the fiber optics bundle. This is necessary to allow the summing of signals from all channels into a single set of bins. The early channel is delayed by approximately 7/64th of the total scan time; each successive channel being delayed by 1/64th scan time less, until the last channel, which is not delayed. The precise delay for each channel is determined during system calibration and results in exact alignment of all eight channels.

The delay is implemented with an address counter and a RAM chip (see Figure 24). The starting address value is held by the 74HCT574 registers, which are mapped as output ports on the CAMAC bus. The address counters count up until the terminal count is reached, then the ripple carry output (RCO*), after being reclocked, causes the counters to be synchronously reloaded with the start value.

At each memory address the contents of the RAM are read and latched by the RAM shift register input register (see next section), then the value of the edge detector output is written into the same location. The counters subsequently are incremented to the next address. The next time the counters access this RAM address, the value is read out and a new value is written. This causes a delay

of the data equal to the period at which each address is accessed; the delay being equal to the period of the counter rolling over and reloading. The starting value registers are loaded by the software with a value of 65536 minus the desired delay.

Shift register: The 74LS273 (U84) input register provides two functions (see Figure 25). First, the output of the programmable delays are latched to provide a stable signal to the RAM shift register. Second, the clear function of the register allows a means of clearing the RAM. This is done while the contents of the register are being read out. The address counter operates exactly like the programmable delay circuit discussed above.

Two banks of RAM provide a programmable delay of the 8 bit wide data from the channel front ends. This gives the effective result of a shift register with taps at the beginning, middle, and end. The three taps are used to determine the image profile center and the middle and end taps are used during read out of the profile. At each address the contents of the RAM are read and latched. The first, or left, bank output is latched into the 74ALS575 (U85) register and the left data output register. The same data is loaded into both registers. The 74LS575 acts as intermediate storage of the left bank output which is written into the right bank during the write phase of the cycle. The output register provides the data to the pulse counter. The second, or right, bank output is latched by the right data output register. After reading, new data values are written into each bank at the same address. The left bank receives its input from the 74LS273 input register, via the 74HCT574 register acting as a tristate buffer. The right bank receives its input from the 74ALS575, which also provides the means to clear out the right bank. After writing the new data the address counters are incremented. This process continues as long as SCLK is enabled. Clearing of the shift register memory is done while the data is being read out. The input register and the 74LS575 intermediate storage registers are held in the clear state by the CHXCLR* signal so zero values are written to all memory location being read out. The memory is then ready to acquire the next image scan.

The output registers are the middle and end taps on the shift register. The input register is the beginning tap of the shift register. All three registers provide a stable value to the pulse accumulator representing the pulse samples for the 8 channels.

Pulse counter/center finder: The eight channels are scanned sequentially by the 74ALS151 data selectors (U86, U87, U88) (see Figure 26). A 3 bit counter operating at 9 times the sample clock rate is implemented in a 22V10 programmable logic device (U90) to generate the address for the data selectors (see Appendix E for program code). All channels are addressed during one sample clock period. The data appearing at the output of the three data selectors represents the pulse state at the left bank input, left bank output, and right bank output of the shift register.

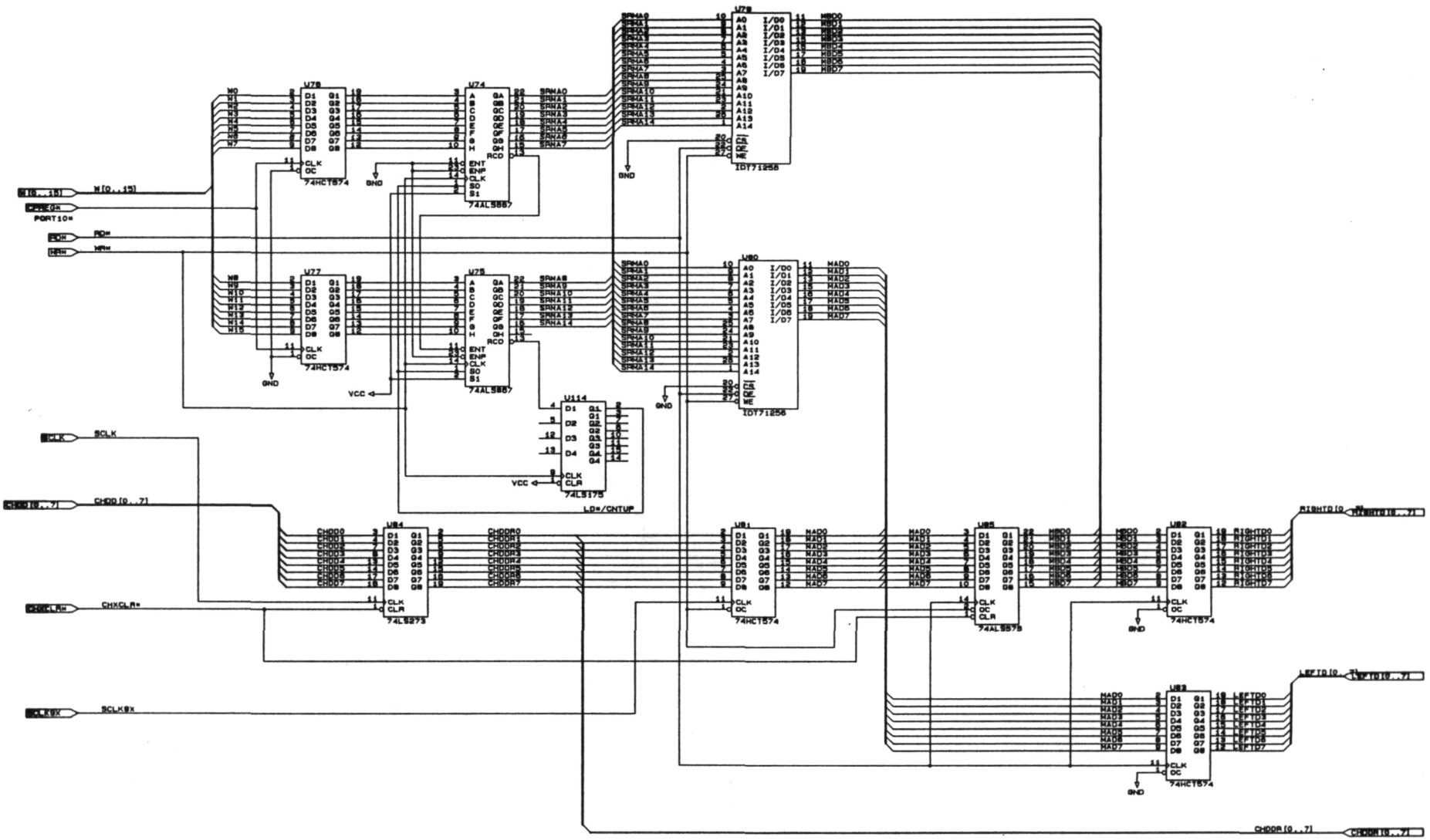


Figure 25. Shift register schematic

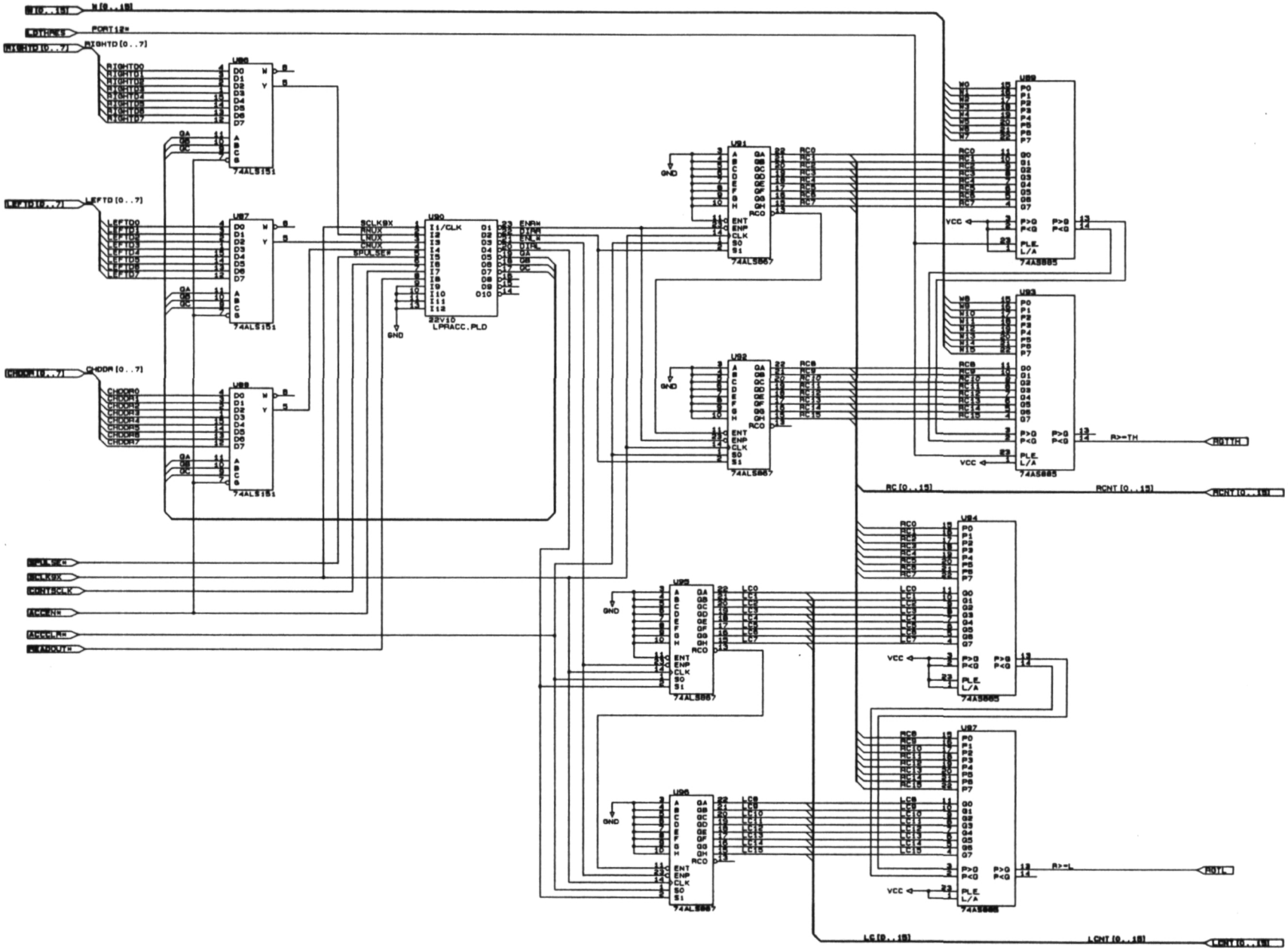


Figure 26. Pulse counter schematic

A comparison of the data selector outputs is made to determine data entering and leaving the left and right shift register banks. When a pulse appears at the input to either shift register bank the logic enables the pulse counters to count in the up direction. When a pulse appears at the output of either bank the pulse counters are directed to count down. When no pulse appears at either input or output, or a pulse is entering and leaving the shift register bank simultaneously, the counters are not enabled to count because there is no net change in the pulses contained in the shift register. The left bank is enabled by the exclusive ORing of the left bank input with the left bank output. When the sample entering and leaving is not the same a change to the count is needed. The counter direction is determined by the level of the left bank input. The direction is up if the input is active, the direction is down if the input is not active. The right bank is enabled by the exclusive ORing of the left bank output (which is also the right bank input) and the right bank output. The logic is the same as for the left bank.

Pulse counts contained within the two banks of the shift register are maintained by the pulse counters. A pulse is a shift register sample location which contains a data value of one; no pulse is represented by a value of zero. The counters have a continuously running clock input of nine times the sample clock frequency (SCLK9X). The counters count in the up or down direction under control of the counter logic described above. A decision is made at each 9X clock based on the data appearing at the data selector outputs. Eight such decisions are made during each sample clock corresponding to the eight channels. Data from all eight channels is summed by the pulse counters. The counters run from a 9X clock so that all eight channels can be counted before the next sample clock rising edge. During the ninth clock, the comparators settle and the control logic makes a decision at the next sample clock rising edge about threshold and centering based upon the results of the previous sample. During the image acquisition mode the pulse counters are used to count the pulses contained in the entire left and right shift register banks. During the readout mode the pulse counter counts the pulses in a bin. At the beginning of the readout sequence the counters are reset, the control logic enables the clock to the shift register for a count equal to the number of samples in one bin. After the shift register is stopped, the counter values are loaded into the FIFO buffer. The counters are reset again, the shift register is enabled and pulses are counted for the duration of another bin. This process repeats until the entire shift register is read out. For proper image centering to occur, the pulse counters must always reflect the true count of pulses in the shift register. This is accomplished by clearing the memory during readout and clearing the pulse counters before acquiring the next image.

To avoid acquiring an image which contains only noise, the right bank pulse counter must reach a software programmed value before enabling the control logic to acknowledge that an image is pre-

sent. The 74AS885 threshold comparators (U89, U93, U94, U97) contain a latch on the P inputs and are programmed by software with a 16 bit value during instrument configuration. Two 8 bit devices are cascaded to provide a 16 bit comparison. When the right bank pulse counter value connected to the Q inputs equals or exceeds the P latch value the R >= TH signal is asserted high. This threshold is different from the discriminator threshold. The discriminator threshold allows a digital pulse to be generated only by those PMT pulses which exceeds amplitude. The pulse counter threshold causes image profiles with insufficient total pulse counts to be ignored.

Centering of the waveform ideally is accomplished by stopping the shift register clock when the pulse count in the left and right banks are equal. However, because an odd number of pulses may be contained within the shift register, exact equality is not always possible. Therefore image center is detected when the right bank pulse count is greater than or equal to the left bank pulse count. The P latch is tied high placing it in the transparent mode so the P>Q signal (actually P>=Q, labeled R>=L) always represents the comparison of the two pulse counters. When image center is detected the right bank may be more than one count greater than the left bank because the centering decision is made after scanning through all 8 channels. More than one channel may contain a pulse during each sample.

Clock signal generator: The control logic for each channel can be divided into several categories; clock generation, bits per bin counter, bin counter and state machine control. Several of these functions have been implemented utilizing programmable logic devices. Clock signals are derived from a 5.76 MHz oscillator (U113) (see Figure 27). A programmable logic device (U111) implements a 4 bit counter (see Appendix E for program code). This, in turn, divides the oscillator frequency down to the 640 kHz which is used as the sample clock. A 4 bit divider is used to divide by nine. The counter bits generate the continuous sample clock signal, together with a single 9X clock wide pulse (SPULSE*).

Outputs from the PLD are read clock (RD*), write clock (WR*), sample clock (SCLK), continuous sample clock (CONTSCLK), and sample clock pulse (SPULSE*). RD* and SCLK are identical in the current circuit implementation. WR* is the opposite polarity of RD*. SPULSE* occurs at the end of each SCLK cycle, the rising edge of SPULSE* coincides with the rising edge of SCLK. All of these signal are reclocked by a 74HCT574 (U112) using SCLK9X to provide buffering. SCLK, RD*, and WR* are synchronously enabled and disabled by the SCLKEN signal generated by the control logic. WR1*, WR2* and WR3* are separately buffered versions of WR* to distribute the signal with acceptable signal loading.

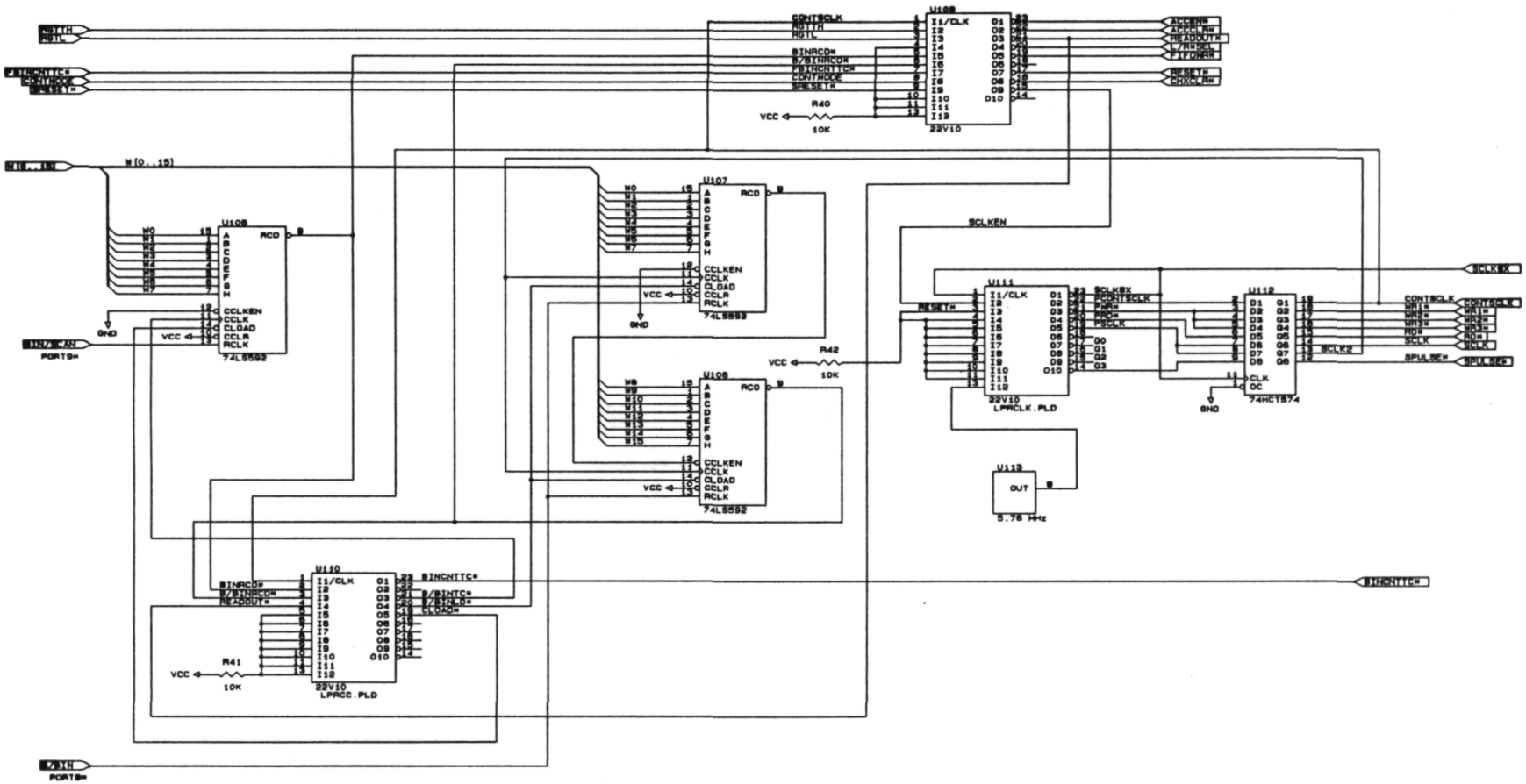


Figure 27. Clock signal generator, bits per bin counter, bin counter, and state machine controller schematic

During each SCLK period the pulse prescaler output is sampled and loaded into the shift register. A given sample may contain a pulse, represented by a one bit being stored, or no pulse, represented by a zero bit being stored. The clock rate of 640 KHz over samples the randomly spaced PMT pulses to reduce to an acceptable level the probability for two PMT pulses occurring during one sample period. The maximum average raw PMT pulse rate from the discriminator is about 1 Mega-pulse per second (MPPS). The prescaler reduces this to 100 kPPS maximum average. Pulse rates above this level significantly increase the probability for two pulses occurring during one sample; thus losing one of the pulses and contributing to non-linearity.

Bits per bin counter: During readout of the shift register memory, data is divided into bins. Each bin contains an equal number of samples. The contents of each bin are accumulated by the pulse counters. When the bits per bin counter indicates that all the samples in the bin have been accumulated, the next bin is read. Two 74LS592 8-bit counters (U107, U108) implement a 16 bit counter which counts in the up direction until the terminal count of 65281 is reached (see Figure 27). Because a cascading scheme is used, the RCO* output asserts when the high byte counter reaches its terminal count of 255, even though the low byte counter is at 0. Therefore, the preset register contained in the 74LS592 is loaded with 65536 minus 255 minus the desired bits per bin count.

Bin counter: The bin counter keeps count of the number of bins read out. The programmed bits per bin count multiplied by the programmed bin count determines the amount of shift register memory read during read out and equals the shift register length programmed.

State machine controller: Figure 28 shows the state transition diagram of the control logic for the signal processing electronics. This logic is implemented in a 22V10 programmable logic device (U109) as a state machine (see Figure 27) (see Appendix E for program code). At each rising clock edge, a state change occurs which is based both upon the input signals and upon the current state. Some states transition unconditionally to the next state, others require an event to occur first. The specific states are defined in Table VII.

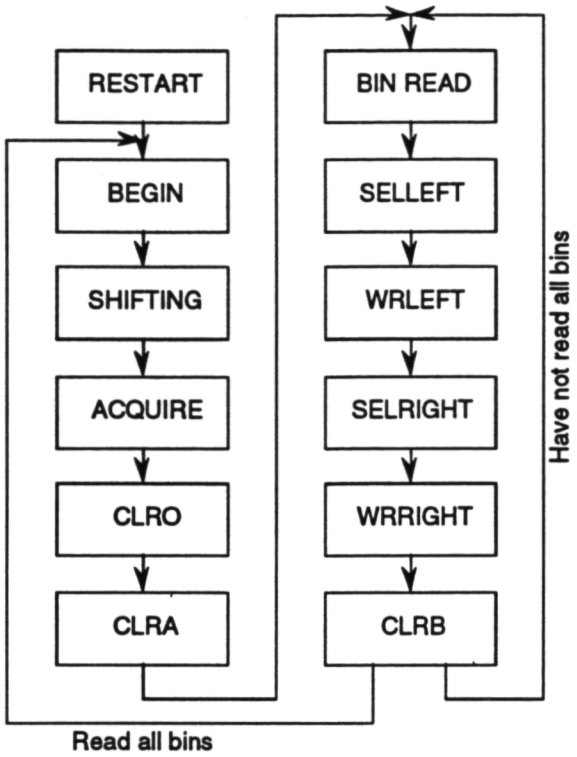


Figure 28. State transition diagram

Table VII. Digital board states

RESTART	Where the sequence starts after a reset of the PLD
BEGIN	Wait for continuous mode to be asserted Shift register is being cleared
SHIFTING	Wait for right bank count to cross threshold Shift register is enabled
ACQUIRE	Wait for image center to be seen
CLR0	Let clock run one more time to preload output registers
CLRA	Clear the pulse counters and FIFO
BINREAD	Enable clock and wait for bin to be read
SELLEFT	Stop the clock and select the left counter output
WRLEFT	Write the left counter value to the FIFO
SELRIGHT	Select the right counter output
WRRIGHT	Write the right counter value to the FIFO
CLRB	Clear the pulse counters

FIFO buffer: To reduce the number of FIFO buffers required, the left and right pulse counter values are interleaved (multiplexed) and stored in a single 16-bit wide FIFO (implemented with two IDT7201 9-bit wide FIFOs (U102, U103)). De-interleaving of the bin values is done by the com-

puter as part of the data processing and display. The left and right pulse counters input to the 74LS157 multiplexers (U98, U99, U100, U101) (see Figure 29). The control logic selects the left bank counter, clocks the FIFO write line, then selects the right bank counter and clocks the FIFO write line again. This process repeats for all bins read out. At the completion of the readout the FIFOs contain all the bin values. When the FIFO is read but is empty, the FIFO internally inhibits the read operation and the outputs remain in a tristate condition. A pull up resistor on the most significant bit causes a value of 32768 or greater to be output on the CAMAC read bus if the FIFOs are read when they are empty. This provides the means for a validity check on the data. If the first an last bin values read in are less than 32768, then the entire array is valid. The FIFO outputs drive 74HCT576 inverting registers which latch the FIFO data and provide the inverting data buffers required by the CAMAC bus.

CAMAC bus timing requires that read data be valid before the leading edge of the strobe 1 (S1) signal, but no irreversible action is to be taken before the S1 signal. Reading of the FIFO is an irreversible action because once the data is read it is lost. If initiation of the FIFO read operation were based only upon the board select (N), function code (F1 to F16), and address (A1 to A8) spurious reads could occur while these signals are settling. Therefore, the S1 signal is used to qualify the FIFO read clock. All signals must be valid during S1. But, if S1 is used to initiate a read operation, the data can not possibly be valid on the leading edge of S1. These opposing requirements are reconciled by using a register on the FIFO output. The FIFO read clock trailing (rising) edge clocks the data appearing on the FIFO output at the end of each read cycle. The next time the FIFO is read, the previously latched value is read and a new value is latched into the register.

As a result of the added pipeline delay, an extra read cycle is needed to get the register loaded the first time. The first read cycle returns invalid data and must be discarded by the software. The last read cycle clocks invalid data into the register as the last valid data is being read out (because the FIFO is empty at this point so no FIFO read actually occurs.)

CAMAC interface: All CAMAC bus signals are active-low levels, including the read and write data lines. Consequently, inverting buffers are used to receive and drive the signals; thereby allowing active high levels to be used in the digital electronics. All inputs are loaded by one LS TTL gate or by a PLD input.

CAMAC port decoding is performed by a 22V10 programmable logic device (U6) and a 74LS138 decoder (U7) (see Figure 30). The PLD decodes the CAMAC bus function codes, address, and control signals to generate seven write select signals (see Appendix E for program code). The first,

labeled PORT0TO7EN*, is asserted when a CAMAC write is occurring to address 0 through 7, and enables the 138 decoder to generate 8 individual device enable signals for the eight channel front end delay registers. The other select signals comprise individual addresses which are assigned to other programmable functions on the board. The specific addresses and functional codes associated with the I/O ports are listed in Table VIII.

Table VIII. I/O Port Map

Address	Function Code	Port
0	16	Channel 0 delay
1	16	Channel 1 delay
2	16	Channel 2 delay
3	16	Channel 3 delay
4	16	Channel 4 delay
5	16	Channel 5 delay
6	16	Channel 6 delay
7	16	Channel 7 delay
8	16	Bits per bin count
9	16	Bins per scan
10	16	Shift register length
11	16	Reset and Mode control
12	16	Threshold value
13	16	Channel enable
0	0	Read FIFO data

The only read port is the FIFO buffer, which is enabled by the FIFOWR* signal. READOE* is used to enable the tri-state drivers to output the data on to the CAMAC read bus. Write port 11 is the mode control port used both to perform a software reset of the control logic and to select the continuous running or single scan mode. Specific information required to program the system is presented in Appendix E.

Power Supply: The power supply schematic associated with the signal processing electronics is shown in Figure 31.

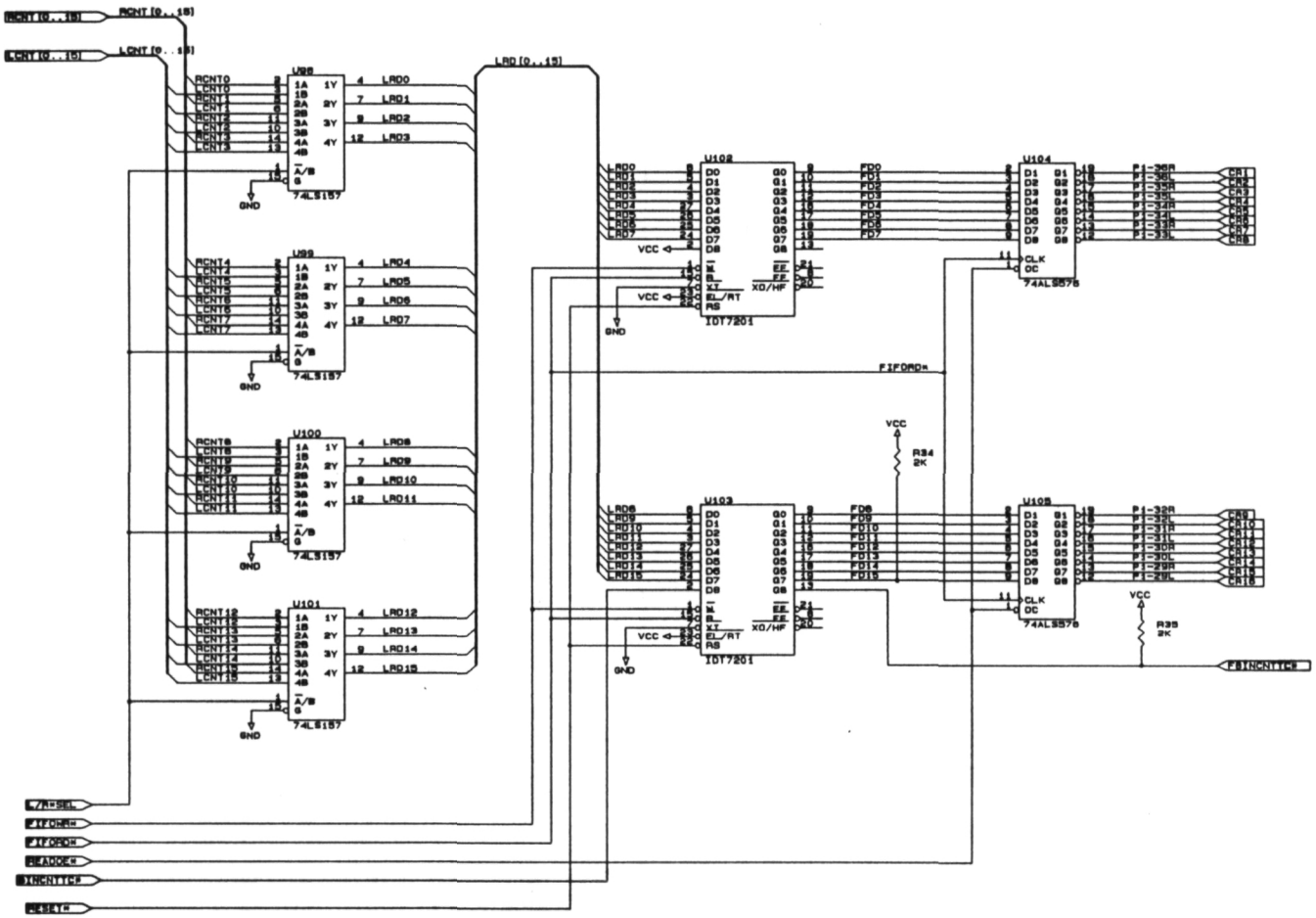


Figure 29. FIFO output schematic

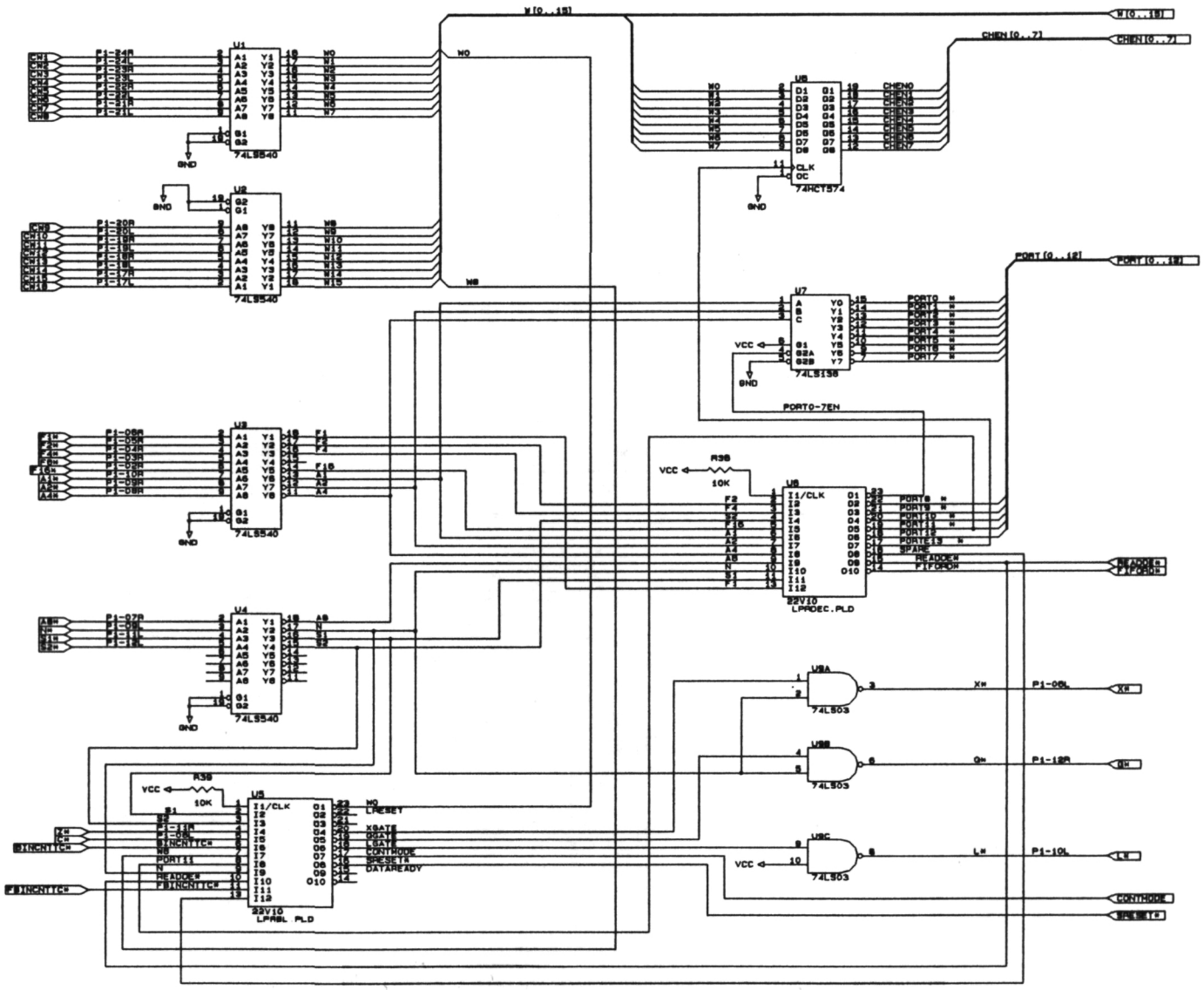


Figure 30. CAMAC interface schematic

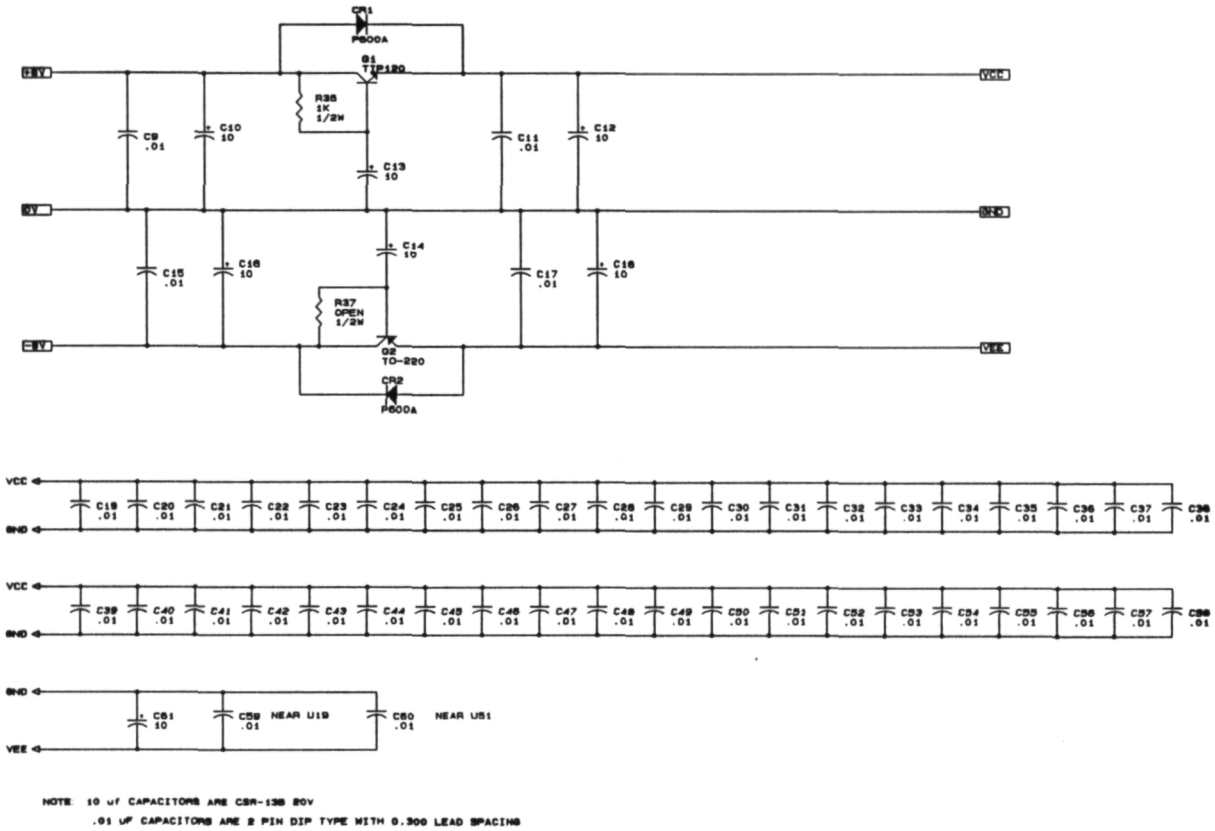


Figure 31. Power supply schematic

3. Software

For the Phase II program, the host computer is a 16-MHz Macintosh IIcx with a 32-bit Motorola 68030 microprocessor, a 68882 math coprocessor to allow real-time data reduction and 8 MBytes of RAM. Display of data is via an 8-bit color monitor, along with a dot-matrix printer for the provision of hard copy output. Communication between the head unit, digital electronics and the computer is via a National Instruments GPIB board within the computer, the computers' RS-232 serial port, and the computers' Small Computers Systems Interface (SCSI) port. A 7-bit video digitizing board, produced by Aapp's Corporation, situated internal to the CPU permits real time display of the Cohu CCD video image upon the computer monitor.

CAMAC compatible hardware requires a controller which is resident on the hardware bus to issue commands to each of the individual modules within the CAMAC crate and to transfer data between the host computer and the modules. The SCSI CAMAC crate controller selected for this function is compatible with existing I/O ports on the Macintosh. (The SCSI specification defines a standard approach for interfacing peripheral devices to one or more host computers.)

Software for control of the head unit and signal processing electronics is in the form of compiled Pascal, written using Symantec's Think Pascal 4.0 for the Macintosh (see Appendix H for full software code)*. This software supports all of the required system operating features described previously. These features are fully operator controllable and include image rotation for control of the scan axis, both coarse and fine focus, selection of image attenuation and magnification, control of image scanning speed, full control of all functions of the signal processing boards by means of the CAMAC/SCSI interface, and real time, unattended data acquisition.

The graphical interface for the control software is shown in Figure 32. All required commands are accessed and sent either via pull-down menus or by elements within the main software window. To the left of the screen is a histogram plot, which presents the most recent image data received from the signal processing boards. The plot shows the total number of counts in a bin versus bin number. A real-time digitized video image from the CCD camera is seen in the upper right hand corner. This image provides feedback to the user and allows for centering and focusing of the object under observation.

Control of image centration is by means of the scrollbars which are to the right and the bottom of the digitized video image. The horizontal scroll bar controls the y-axis translation stage to provide

* The most current version of the Digital Profiler software is "Digital Profiler1 10/7/91".

horizontal translation. Meanwhile, the vertical scrollbar rotates the microscope objective wheel to provide vertical translation of the image. Focus is controlled by the two icons that are below the bottom scroll bar. By clicking upon either icon, the z-axis translation stage is moved a predetermined number of steps forwards or backwards along the optical axis.

At the bottom of the screen is a status bar which gives information on the current object being observed, the number of scans acquired, the number of scans per second, the current magnification and attenuation, the total number of counts in all bins, the peak number of counts in any single bin, and the peak number of pulses per second.

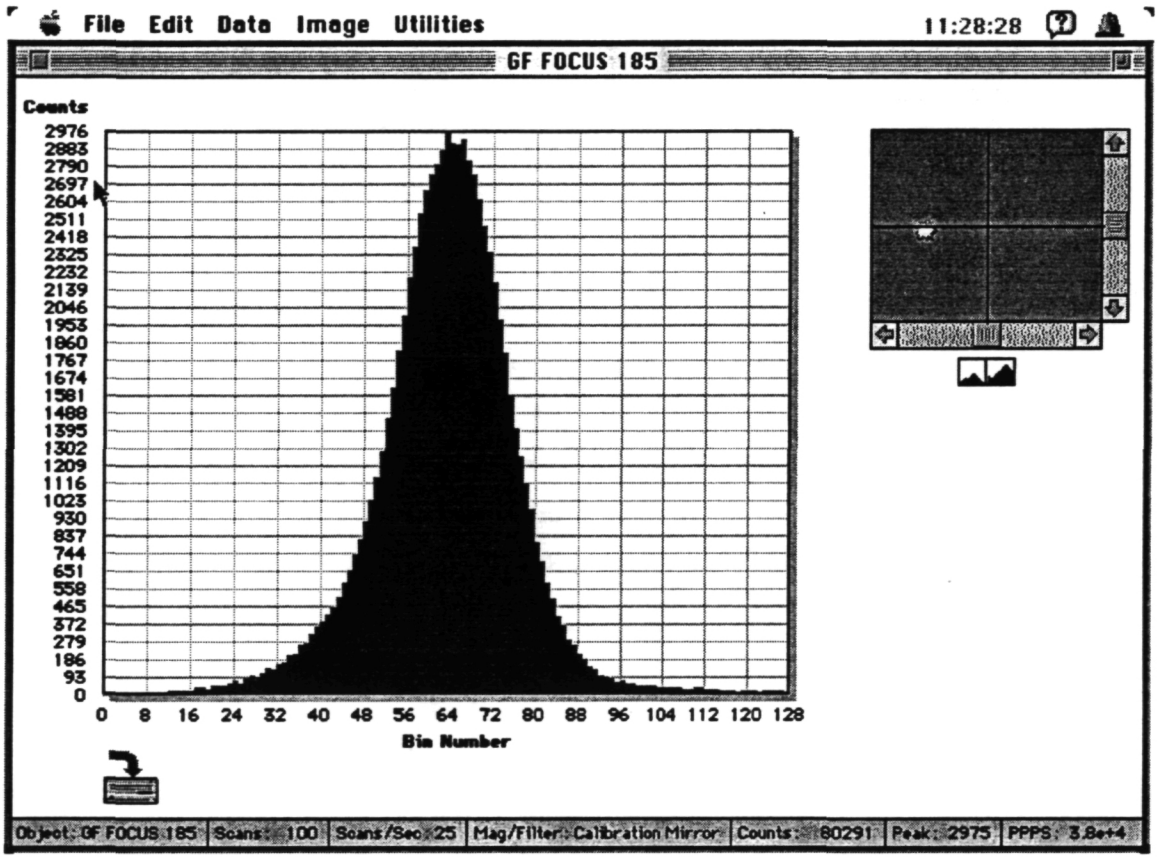


Figure 32. Digital Profiler software interface

There are five main pull down menus from which most commands are accessed; these being File, Edit, Data, Image, and Utilities. Each main menu includes a number of additional sub-menus. Descriptions of the functions performed by each sub-menu are presented in the ensuing subsections.

File menu: The sub-menus under the File menu provide for printing and saving of image data to disk. The "New" command is not used. "Open..." allows a previously saved data set to be re-

displayed in the histogram window. The "Close" command is not used. "Save..." prompts the user for a file name under which to save to disk the current data set together with all information entered using the "Object Info..." command under the Utilities menu. "Save As..." allows for saving of the current data under a different file name and/or in a different directory. Files generated using the "Save..." and "Save As..." are only for use by the Digital Profiler software. For use in other applications, the "Export as Text" command must be used. "Export as Text" saves the current data set as a tab-delimited ASCII text file; this format being necessary for post-processing of the data. "Export as PICT" saves a PICT format file (a Macintosh graphics format specification) of the current histogram for importation into other applications. "Page Setup..." presents various printing options. "Print..." prints the current histogram to the currently selected printer. "Quit" exits the Digital Profiler software.

Edit menu: The commands listed under the Edit menu are not used by the Digital Profiler software. They are provided for use by other applications.

Data menu: Control of all user programmable attributes of the signal processing electronics is performed by the commands located under the Data menu. The "Acquire Data" instructs the signal processing electronics to obtain the programmed number of scans and to send the resulting data to the computer. "Clear Data" resets the computers data array to all zeros and removes the histogram from the plot on screen. "Real Time" issues a series of "Acquire Data" and "Save" commands to allow for the acquisition and saving to disk of a number of data sets with no user interaction required. The series of data sets acquired are saved as a sequential set of files where the file name consists of the Object Name entered using the "Object Info..." command under the Utilities menu followed by the current scan number; i.e. And-Gamma1, And-Gamma2, And-Gamma3, etc. "Number of Scans..." is used to set the number of image scans that are accumulated by the signal processing boards. "Number of Bins..." sets the number of bins into which the image will be divided (normally 128). "Number of Boards..." determines the number of digital boards which are used for gathering of data (normally 8). "Threshold..." sets the programmable threshold level on the Phillips Scientific discriminators modules.

"Enable/Disable Channel..." sends either an enable or a disable command to individual channels on the signal processing boards. This function is useful for isolating channels when determining the location of problem areas, such as bad amplifier channels or damaged signal cables. By properly modifying the Digital Profiler software (in the DPMain.p unit) it is possible to have problem channels automatically disabled when the software is launched. "Plot Histogram" replots the histogram plot using the most recently acquired data set. "Histogram Color" allows the user to choose the color of the data plot.

Image menu: Control of optical elements within the head unit is via commands under the Image menu. Additional control is provided by elements within the main software window, as described previously. "Microscope Objective" presents a hierarchical sub-menu allowing for selection of image magnification and attenuation or for selection of the calibration mirror which provides an image on the CCD. "Start Scan Mirror" initiates polygonal scan mirror rotation at the selected rotational velocity. "Scan Speed" displays a hierarchical sub-menu for selecting the number of scans per second; hence the mirrors' rotational velocity. "Stop Scan Mirror" halts rotation of the scanning mirror. "Stage Resolution..." provides selection of the number of steps that each translation and rotation stage moves when the appropriate control is activated in the main window. "Center Stages" moves all translation and rotation stages to their nominal center positions. "Calibrate Stages..." is used to input new values for the nominal stage positions. These values remain in effect until power is removed from the Klinger MC4 controller.

Utilities menu: The Utilities menu contains various commands which are useful for general system parameter information and debugging purposes. "Object Info..." is selected in order to input information concerning the object under observation, including object name, R.A., Dec, and the observatory at which testing is being performed. This information is saved along with profile data. "Parameter Info..." gives feedback regarding the currently programmed values of scan rate, number of bins, shift register length, and digital board threshold. "Deskew..." calculates the variable delay on each channel to correct for the displacement in time of the scanned image along the fiber bundle. These calculated values are used to set the programmable delay on the digital boards. "Send GPIB Command..." allows for sending of commands to the National Instruments GPIB board which is internal to the computer. All the translation and rotation stepper motor driven stages are controlled via the GPIB interface. This command is useful for debugging purposes and for sending commands which are not directly implemented in software. "Send Serial Command..." allows for sending of commands via the Macintosh's RS-232 serial port. The only device controlled via the serial port is the scanning mirror drive motor. This menu command is also useful for debugging purposes.

IV. LABORATORY TESTING

The ensuing discussion of the laboratory test program is divided among six principal topics. These are entitled Experimental technique, Calibration of a temperature stabilized tank, Operation in the absence of turbulence, Uncompensated images in the presence of turbulence, Auto-centered images in the presence of turbulence, and Strehl intensity ration selection of auto-centered images.

1. Experimental Technique

The basic experimental configuration used for laboratory testing consisted of a diffraction-limited monochromatic source, a turbulence simulator to impose dynamic phase fluctuations upon the source, and an image profile sensor; the latter incorporating a means for compensating various types of wavefront aberrations. Arrangement of the optical path of the system was such that various equivalent telescope apertures and turbulent conditions could be simulated. The overall geometry of the test configuration is shown schematically in Figure 33.

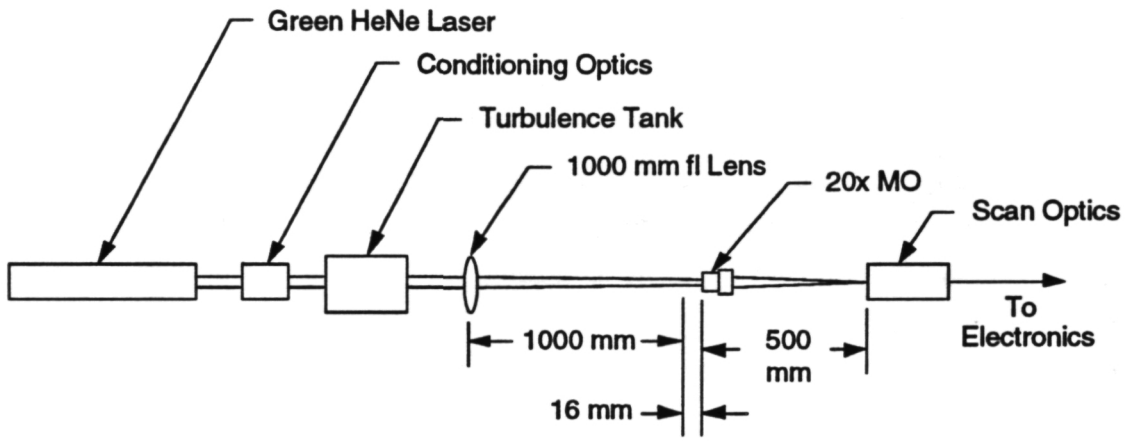


Figure 33. Turbulence test configuration

The optical source comprises a green helium neon laser (wavelength 543.5 nm) with a power of 1.1 milliwatts; the output of which is spatially filtered and expanded to form a collimated beam. By using collimating lenses of different focal lengths, the diameter of the resulting output beam can be selected at will; thereby making it possible to simulate telescope diameters of various sizes.

The statistical properties of the turbulence simulation tank are such that its transverse correlation distance is on the order of 2 mm. Thus, recognizing that the atmospheric correlation distances to be simulated fall in the range of 10-20 centimeters, the beam diameter scale factor ranges from 50:1 to 100:1. Consequently, to simulate a telescope aperture of 1 meter, for example, the beam diameter within the tank is set within the range 10-20 millimeters.

After passage through the turbulence simulator, the beam is refocused by a lens of 1-meter focal length. This results in an image which simulates that of a 1-meter telescope having an equivalent focal length of 100 meters; i.e., with an f-number of 100. Consequently, the resulting diameter of a diffraction-limited image of a point source is equal to 100 times the wavelength of that source. For the green HeNe laser wavelength, the image diameter is about 0.05 millimeter. In contrast, the breadboard image scanning system is designed to work with an image diameter of 3 millimeters. This sixty-to-one disparity in image size is accommodated by using a 20X microscope objective in a manner which provides a back focal distance which is three times its nominal value of 160 millimeters.

The optical path between the turbulence tank and the 1-meter focusing lens is arranged to include two folding mirrors. Also, provision is made for a plane parallel zinc selenide plate to be inserted within the beam at a point immediately ahead of the microscope objective. Multiple internal reflections within the plate provide a low-intensity ghost image of the primary image; thereby permitting the performance of the system to be assessed with respect to low-contrast features within the image. Tilting of the plate allows the angular separation between primary and ghost images to be adjusted in angle from zero to approximately three times the diffraction limit of the system.

Preliminary experiments conducted during Phase II with the breadboard profiler were of three types. First, we operated the system with the optical beam passing through a temperature stabilized tank so as to obtain a baseline image profile in the absence of turbulence. Second, we established the turbulent dependent Strehl intensity probability distribution in the absence of scan selection. This was undertaken at several levels of turbulence and was used to characterize the turbulence generating tank itself. Having established Strehl intensity ratio distributions at a variety of turbulence levels, a series of experiments were conducted at a single level of turbulence. Comparison of data from these experiments established the reproducibility of the system.

Having characterized the test system, we conducted a series of experiments at a nominal level of turbulence; each experiment being designed to evaluate a specific feature of the profiler system. Here, we progressed through a series of four distinct experiments; basing each data set upon a minimum of 1,000 scans. First, we recharacterized the system in the absence of turbulence; thereby establishing a quality of performance against which subsequent features of the system could be judged. Second, using a modest level of turbulence, we determined the nature of the time-averaged point spread function when no features of the system were implemented; i.e., the nature of this function in the absence of scan centration or Strehl intensity ratio selection. Third, we repeated the first experiment with the addition of scan centration only. Finally, we combined scan

centration with Strehl intensity ratio selection; these experiments being conducted with several degrees of stringency concerning the scan selection criterion.

For purposes of discussion, we divide the ensuing discussion of experimental results in accordance with the procedures described above. Thus, we commence with "Calibration of a temperature-stabilized tank"; progressing to "Operation in the absence of turbulence"; etc.

2. Calibration of a Temperature-Stabilized Tank

To evaluate the optical quality of the turbulence tank itself, we compared the Strehl intensity ratio distribution of the quiescent tank with a distribution obtained when the tank was removed from the beam path. For this purpose, we operated the system with an active scan centration system, but in the absence of Strehl intensity ratio selection. A total of 1,000 scans was used in each run; recording the numerical value of the Strehl intensity ratio for each scan. Here, we define the Strehl intensity ratio as the number of counts in the central twelve bins of the scan divided by the total count in all 128 bins. (Note also that the "Strehl intensity ratio" received from a one-dimensional scan is equivalent to the square root of the Strehl intensity ratio of a two-dimensional point spread function.) Upon the accumulation of 1,000 scans, we sorted the data in ascending order by Strehl intensity ratio; thereby providing a Strehl intensity probability distribution.

We found that the Strehl intensity probability distribution for the no-tank condition was similar to that obtained with a quiescent tank in the optical path; both distributions taking the form of the uppermost curve in Figure 34. This result indicates that the optical quality of the tank itself is such that it contributed negligible errors to the experiments.

Having established a baseline in the absence of turbulence, we performed a similar series of experiments at a variety of power inputs to the tank. As in the no-turbulence case, we accumulated a sequence of 1,000 scans for each condition; recording and ordering the Strehl intensity ratios so as to produce probability distributions. Distributions associated with tank heater power inputs of approximately 32 watts, 64 watts, and 128 watts are included for comparative purposes in Figure 34.

As expected, we see that the mean value of the intensity ratio declines as the level of turbulence is increased. Commencing with a quiescent value of 0.87, we find that, with the introduction of 32 watts to the tank, the mean value of the ratio drops to 0.65. Upon successive doublings of the input power to 64 watts and 128 watts, the mean ratio falls to 0.5 and 0.35, respectively. In terms of a two-dimensional Strehl intensity ratio, the lowest of these figures represents a value of about 0.16.

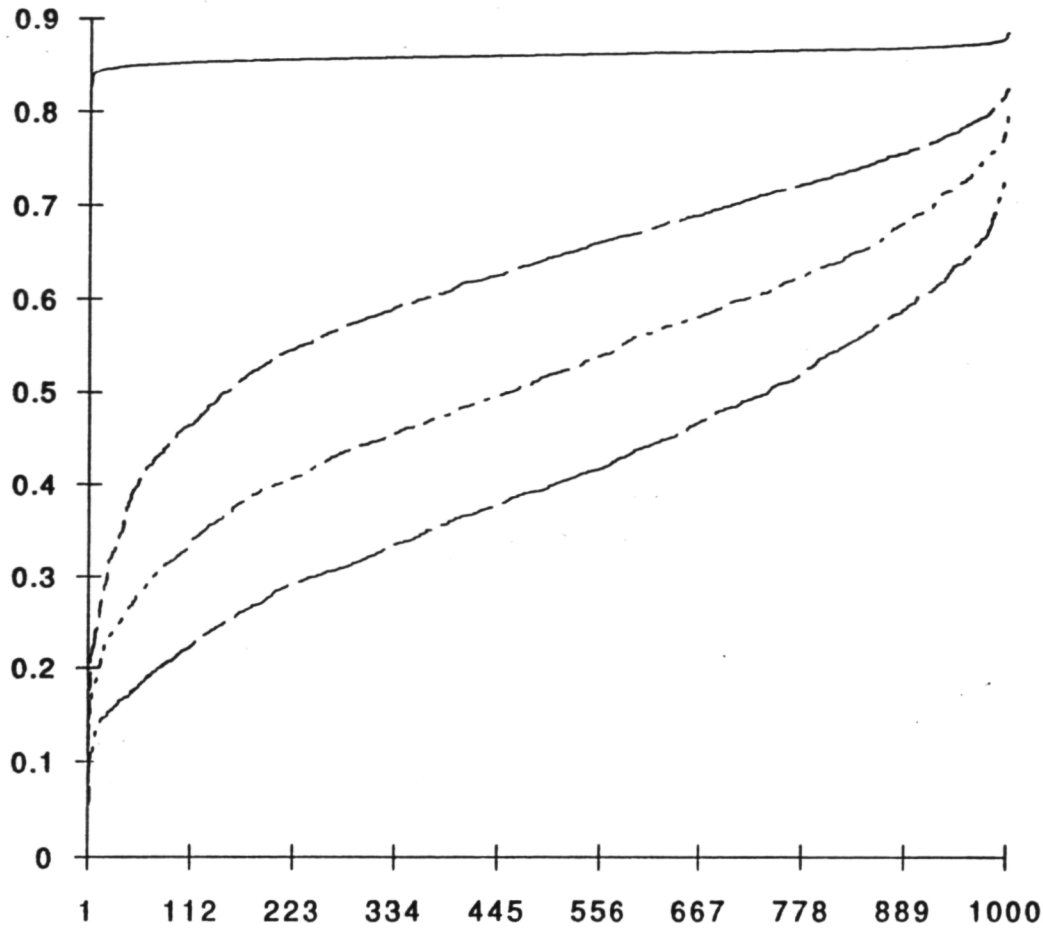


Figure 34. Probability distribution of Strehl intensity ratio

Of greater significance in the context of the profiler test program, we observed that, despite the progressive reduction in mean Strehl intensity ratio, the peak value of the ratio (given by the points at the extreme right-hand end of each plot) is not greatly affected by the increase in turbulence. Rather, the fraction of scans which exhibit high ratios merely decreases with increasing turbulence. For example, at the 128-watt input level, we still find that some 8% of the scans exhibit an intensity ratio in excess of 0.6. It is these infrequent scans of high ratio which contribute to the image enhancement observed during subsequent experiments.

To establish the reproducibility of our experiments, we compared the results of widely separated data runs conducted with similar input power levels. Such a comparison is presented in Figure 35. Here, in each case, the nominal input power to the tank was 128 watts. However, the mean temperature of the fluid in the tank differed by approximately 10°C (22°C for the first run versus 32°C for the second run). In light of this extreme temperature disparity, combined with the close agree-

ment between probability distributions, we conclude that the reproducibility of experimental conditions is sufficient to place a high degree of confidence in comparative results.

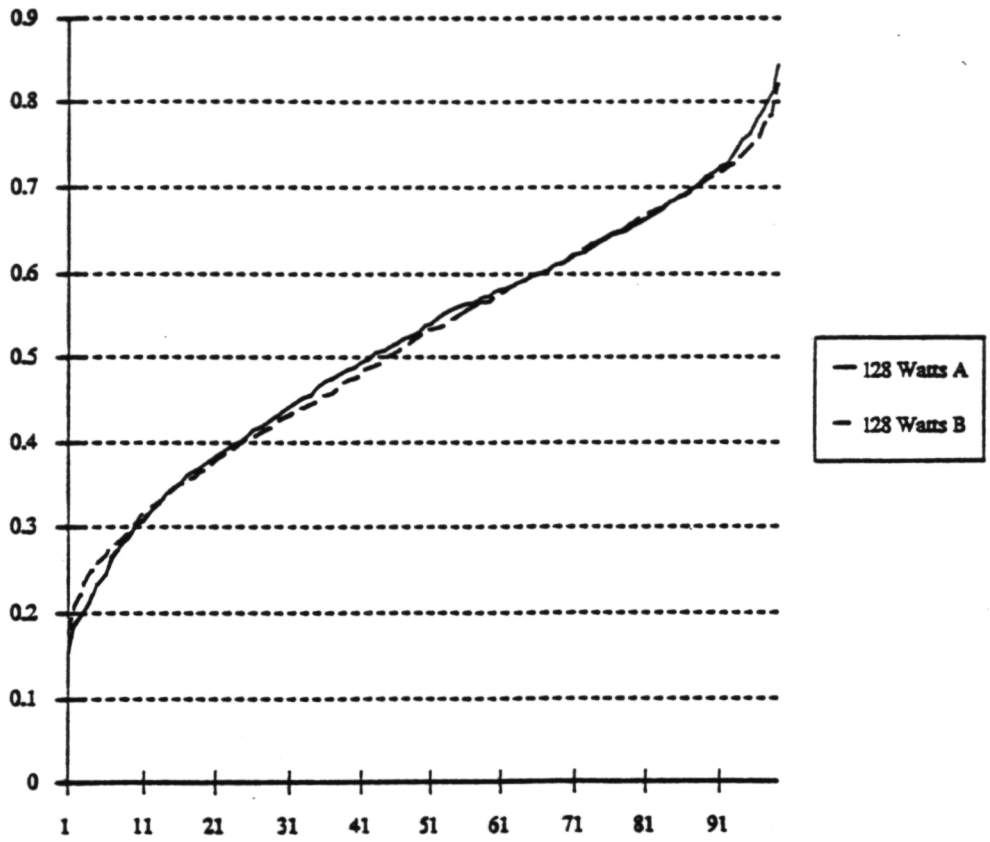


Figure 35. Reproducibility of data under turbulent conditions

3. Operation in the Absence of Turbulence

To establish a baseline for the subsequent evaluation of image profiler features, we inserted a zinc selenide plate, so as to introduce a faint secondary source. Tilt of the plate was adjusted so as to displace the ghost image by approximately twice the diffraction limit of the system. A composite primary/ghost profile based upon 1,000 scans in the absence of turbulence is presented in Figure 36. Here, we note that the ghost image is centered in the vicinity of bin position number 32.

4. Uncompensated Images in the Presence of Turbulence

To examine the nature of uncompensated images in the presence of turbulence, we selected the most severe condition of turbulence associated with our calibration experiments; i.e., that corresponding to the lowermost probability curve presented in Figure 37. The turbulence tank input corresponding to this level is 128 watts. For our first test, we disabled the automatic image centration system; thereby simulating a condition wherein no terms of the turbulence-induced

aberration were corrected. A resulting profile which represents the superposition of 1,000 scans is shown in Figure 34. For comparative purposes we include the original profile for the no-turbulence condition.

As can be seen from an examination of Figure 37, the original intensity ratio of 0.89 is degraded to about 0.23; i.e., is reduced approximately threefold. (In terms of a two-dimensional Strehl intensity ratio, the uncorrected profile exhibits a ratio of about 0.07.) Also, we find that the presence of the dim ghost is masked completely by the enhanced wings of the degraded profile.

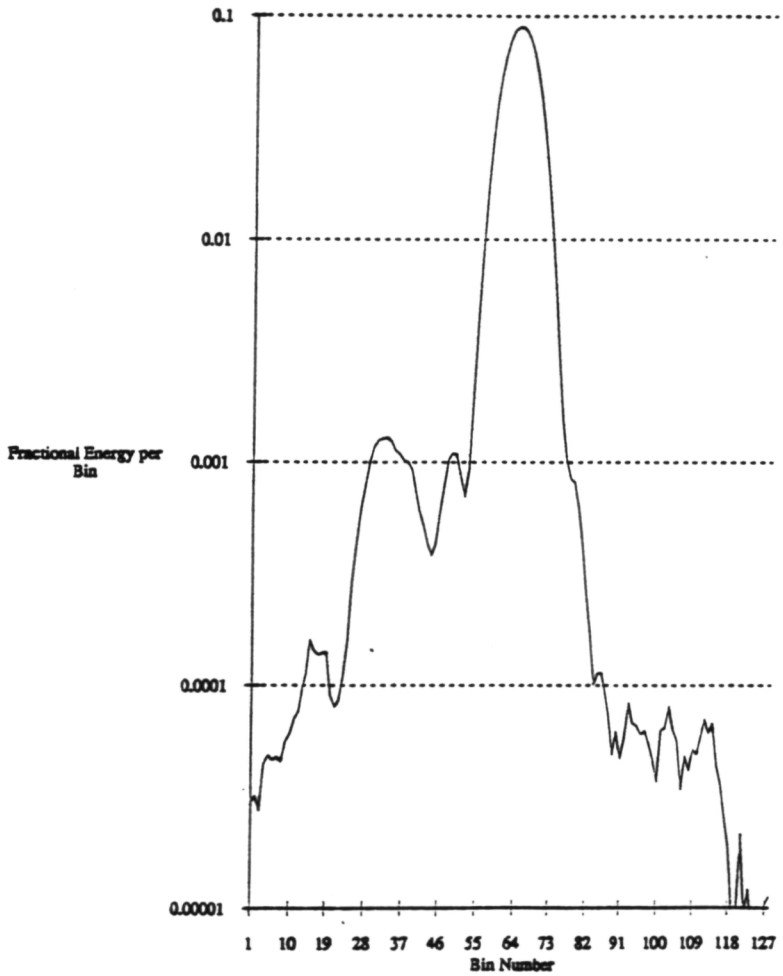


Figure 36. 128-bin profile of quiescent image with dim secondary

5. Auto-Centered Images in the Presence of Turbulence

Upon activating the auto-centration function, we repeated the above experiment. The resulting composite profile is shown in Figure 38. Again, for comparative purposes we include the previous profiles; namely the no-turbulence profile and the no-correction profile.

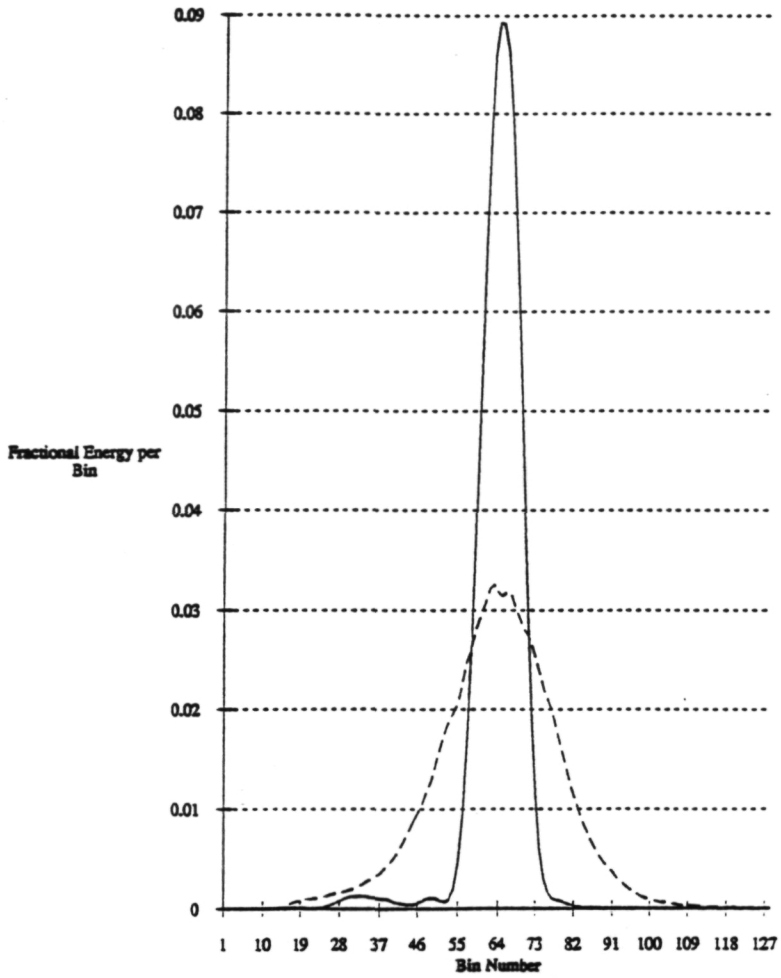


Figure 37. Comparison between quiescent profile and uncorrected profile in the presence of turbulence

Examination of Figure 38 illustrates the dramatic effect of auto-centration; i.e., of removing tilt from the accumulated scans. First, we notice that the intensity ratio has increased from about 0.32 to about 0.45. However, the most significant change in the profile is that associated with the reduction in full-width at half-maximum. In the absence of auto-centration, the full width at half-maximum is approximately 30 bins. On the other hand, when auto-centration is employed, the value falls to 16 bins; i.e., is reduced by a factor of about 2:1.

Despite a dramatic reduction in the width of the central image, it is noted that the wings of the auto-centered profile are still of sufficient magnitude to mask the presence of a secondary source. Thus, despite an overall "sharpening" of the bright features of an image, the implementation of auto-centration alone does little toward improving the visibility of low contrast features within the image.

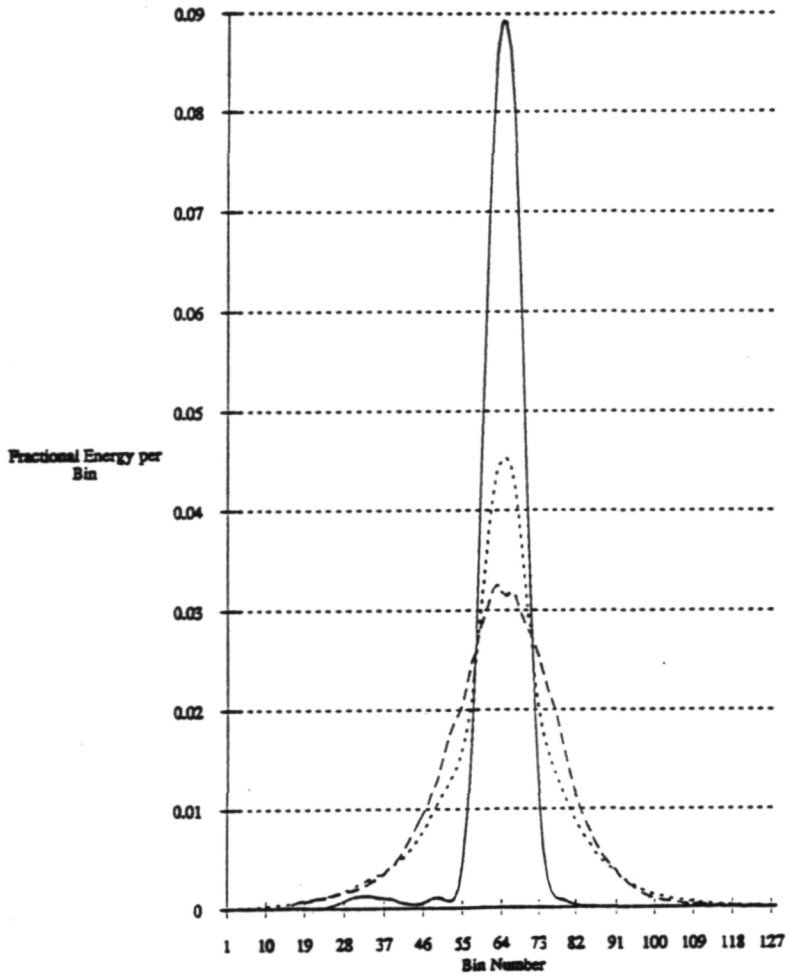


Figure 38. Auto-centered image profile in the presence of turbulence

6. Strehl Intensity Ratio Selection of Auto-Centered Images

The most significant improvement of image quality was achieved when the Strehl intensity ratio selection feature of the image profile system was operated in conjunction with auto-centration. In particular, when the composite profile was restricted to those individual profiles which exhibited ratios in excess of 0.75 (recall that the no-turbulence ratio is about 0.89, while the auto-centered ratio is about 0.45), we obtain the image that is presented in Figure 39. Once again, we include previous plots for comparative purposes.

Examination of Figure 39 shows that the mean intensity ratio for selected scans has been increased dramatically from 0.45 to 0.80. This is to be expected, insofar as we have forced the profile to exhibit a minimum ratio of 0.75. Of greater significance is the extent to which the width of the profile has been reduced to that of the no-turbulence case and the degree to which the wings of the profile have been suppressed. Relative to wing suppression, we find that the companion image is now

clearly identifiable; i.e., we have restored visibility with respect to low contrast features of the image.

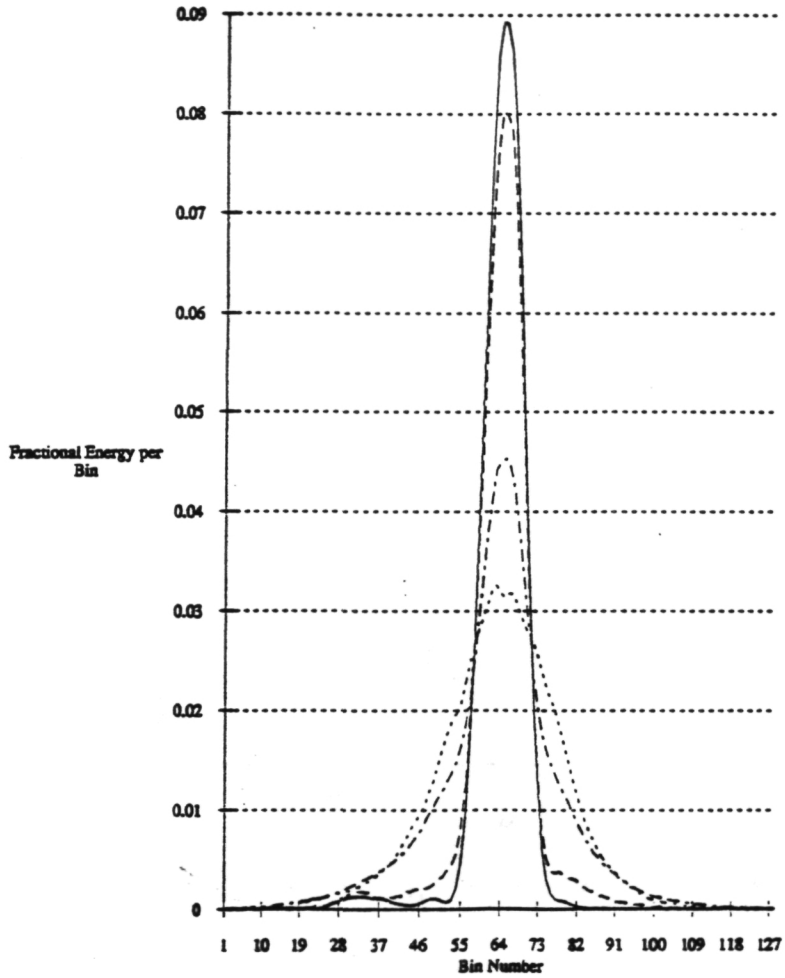


Figure 39. Strehl intensity selected image profile in the presence of turbulence

The extent to which image width matches that of the no-turbulence condition is best illustrated by an examination of Figure 40. Here, we have expanded the vertical scale by a factor of 10 so as to examine the low intensity wings of the image in greater detail. We note that the width of the ratio selected image approximates that of the no-turbulence case down to an intensity level which is about 5 percent of the peak level. In contrast, the non-selected profile fails to approximate such a match even at the 50-percent level.

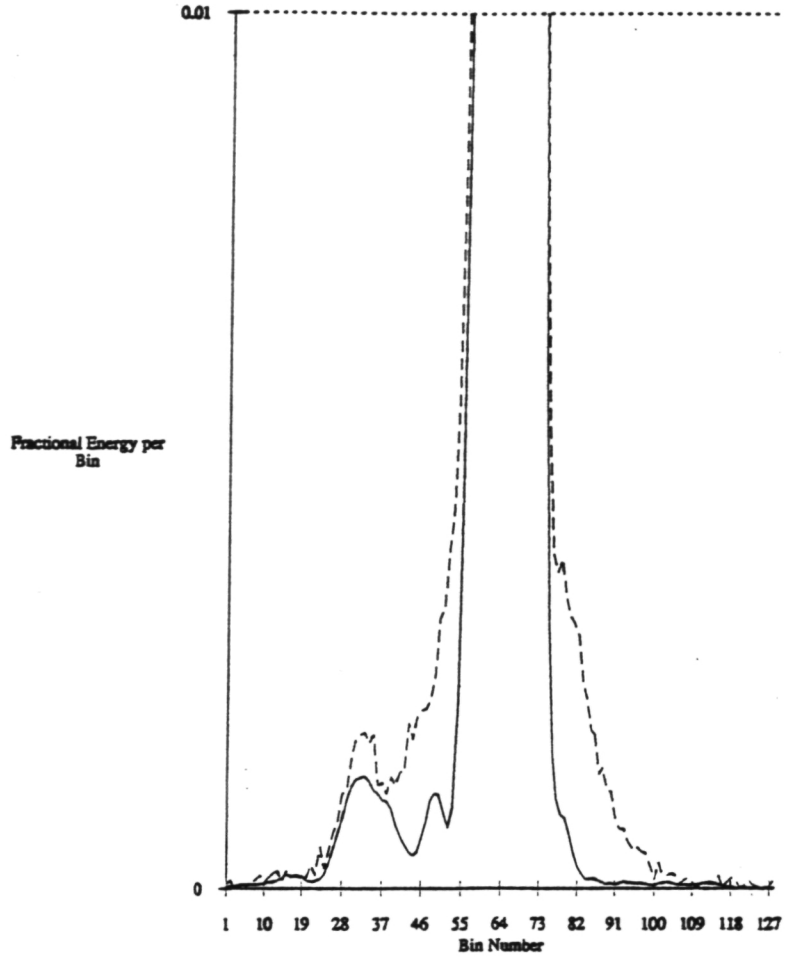


Figure 40. Magnified view of Strehl intensity selected image profile in the presence of turbulence

V. FIELD TEST PROGRAM

It was planned that the field test program should encompass a short shakedown visit to Table Mountain, followed by more extensive visits during which the full potential of the multichannel profiler could be explored. In this regard, the proximity of Laser Power Research to Table Mountain offered considerable program flexibility insofar as the entire profiling system, together with an operator, can be transported from one location to the other via station wagon in the space of a few hours.

Establishing mechanical and electrical interfaces with the 24-inch telescope took place during the detailed design, fabrication and laboratory test phases of the program. This assured that time scheduled on the telescope could be used primarily for testing of the system with respect to selected astronomical objects. Unfortunately, despite this test program philosophy, inclement weather and electromechanical problems associated with the Table Mountain facility caused both the quantity and quality of field test data to fall short of original expectations. Details of the limited observations made during three trips to the mountain are presented below.

1. First Field Test

The first trip to Table Mountain was plagued by a variety of problems. During the first two days of our three day visit, the facility suffered an intermittent electrical brown-out condition. The brown-out prevented operation of the dome, telescope and hydraulic floor, confining observation to a narrow strip of sky and to a few degrees of declination. Electricians dispatched from JPL discovered a corroded circuit breaker. Delayed correction of the problem restricted useful observations to a single night.

During the limited time available, Lyr-Alpha Vega was selected as the preliminary target due to its high brightness. The 24" telescope was commanded to the R.A. and Declination coordinates associated with Lyr-Alpha Vega (R.A.: 18° 34' 57", Dec: 38° 07' 46") and, to our intense satisfaction, the star appeared onscreen at the Macintosh. As mentioned previously, the system software incorporates a routine to select a 45° turning mirror in place of a microscope objective; thereby deflecting a stellar image to the CCD. However, during this early test, it was not possible to position the turning mirror and the CCD surface such that a focused image at the CCD presented a focused image at the fiber bundle.

With the star centered on the optical axis of the profiler, a 5X microscope objective was selected and a number of profiles were acquired of Lyr-Alpha Vega. Figure 41 shows a consolidation of 4 separate profiling runs of the star; these being taken over a 15 minute interval. The profiles are

normalized to the total number of counts in each profile. Scans were obtained at a rate of 25 scans per second; each profile representing the accumulation of 4000 scans.

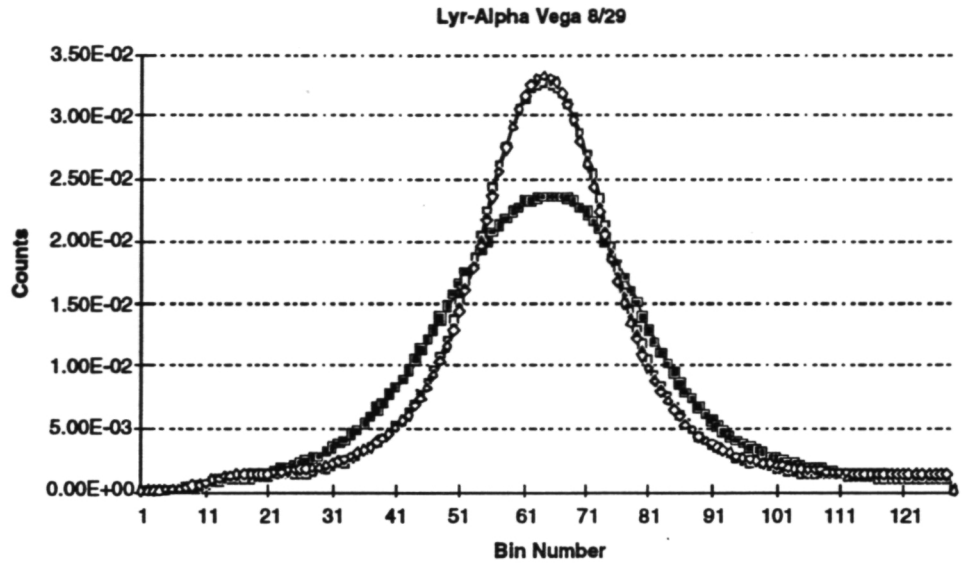


Figure 41. Normalized data profiles of LYR-Alpha Vega

Notice that one profile is broader and of smaller amplitude than the others. The broad scan is the first scan obtained of Lyr-Alpha Vega and illustrates our initial ability to achieve fine focusing. Using the full width at half maximum as a measure of defocus, the position of the 5x objective was adjusted until the narrowest profile was obtained.

Satisfied with the performance of the instrument with a zero magnitude star, attention then was turned to less bright stars. The second candidate for profiling comprised a binary pair in Cygnus, Cyg-Delta. Cyg-Delta is a magnitude 2.87 star with a magnitude 6.5 stellar companion. The angular separation of the pair is 2.2 arcseconds. The profiler was rotated so as to scan the pair in an orientation which maximized detection of the secondary. Figure 42 presents two data sets of Cyg-Delta; each being the accumulation of 1000 scans. Each profile was normalized to the total number of counts in the scan. Here, one notices a shoulder in the left hand portion of the profile. This shoulder was not apparent in the Lyr-Alpha Vega profile. However, with dimmer stars, it appeared consistently. It was suspected that the shoulder was associated with a digital noise problem. The noise level was significant enough to suppress the signal of the secondary in Cyg-Delta. Because of this noise, it also proved difficult to locate stars with magnitudes fainter than $m_v = 5$.

Figure 43 provides a screen shot of Peg-Beta, a bright star in Pegasus. In this figure, each of the problems associated with the first field trip can be seen. The image of the out of focus star (doughnut shaped due to the Cassegrain design of the telescope) is visible in the upper right-hand

corner of the figure. The profile also indicates clearly the high number of noise counts in the wings, together with the curious shoulder in the left-hand side of the profile.

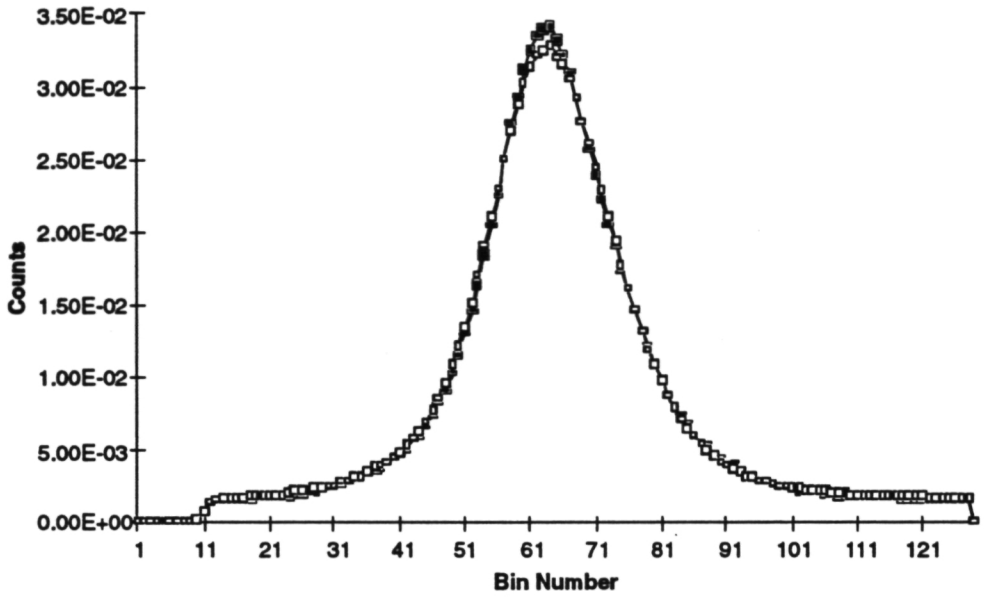


Figure 42. Normalized data profiles of Cyg-Delta

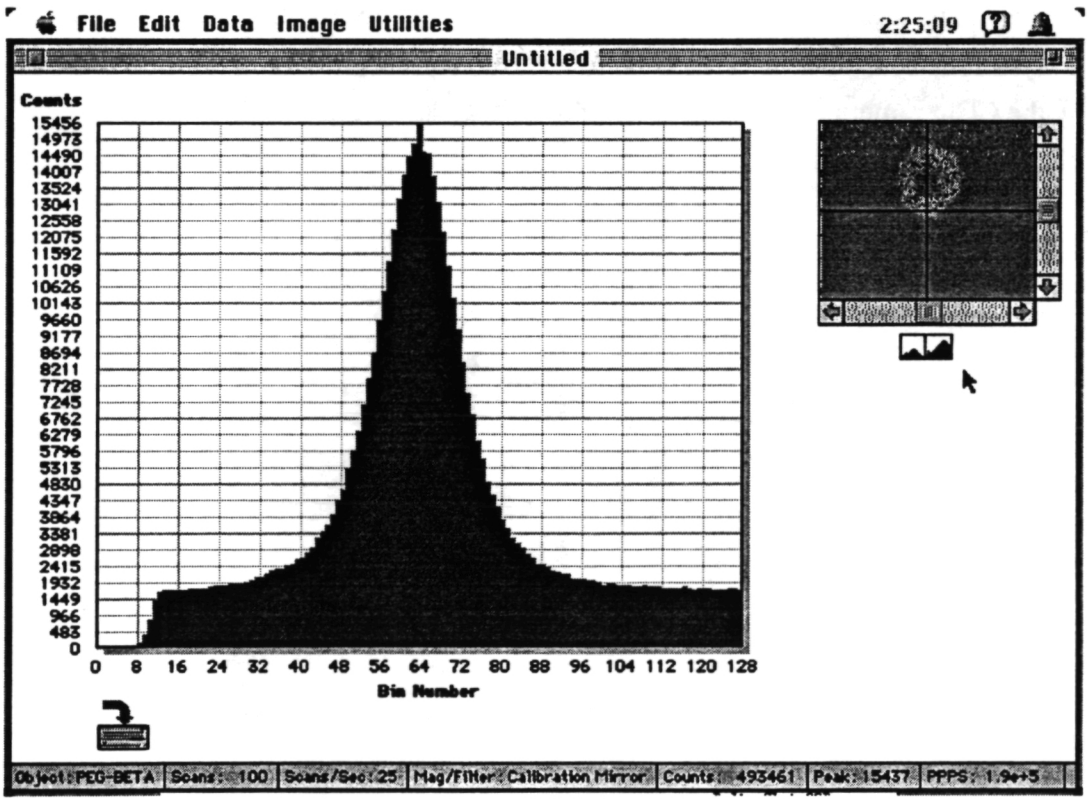


Figure 43. Screen shot of Peg-Beta

2. Second Field Test

Upon returning to San Diego at the conclusion of the first field test, we calibrated the image centration coordinates for the nominal foci of the 5X and 10X microscope objectives. The experimental set-up used a 1.1 mW green helium neon laser (wavelength 543.5 nm) located at one end of a 50 foot optical bench. The beam was focused by a 1 meter focal length lens and allowed to diverge to a diameter of approximately 5 mm. The divergent beam was then refocused by a 200 mm focal length lens and brought to focus at the working distance of the 5X objective (approximately 25 mm). Using the Klinger motion controller digital readout, the focus of the 5X objective was adjusted until a sharp focus was achieved at the fiber bundle. The absolute positions of the optical stages at the point of sharpest focus then were recorded and hard coded into the Digital Profiler software. This procedure was repeated for each microscope objective. With calibration complete, we returned to Table Mountain knowing that an image which appeared focused at the center of the CCD image also would be in focus at the face of the fiber bundle assembly.

During the second field test, we evaluated the performance of each digital electronics board and each of the 64 individual channels. Sources of electrical noise, including loose or damaged connections between the PMT pre-amplifiers and the digital electronics boards and dead or intermittent amplifier channels, were located and corrected. It also was discovered that the CCD camera was the primary source of the previously noted noise problem. Figure 44 is a profile of the noise introduced by the CCD camera in the absence of high voltage to the photomultiplier tubes.

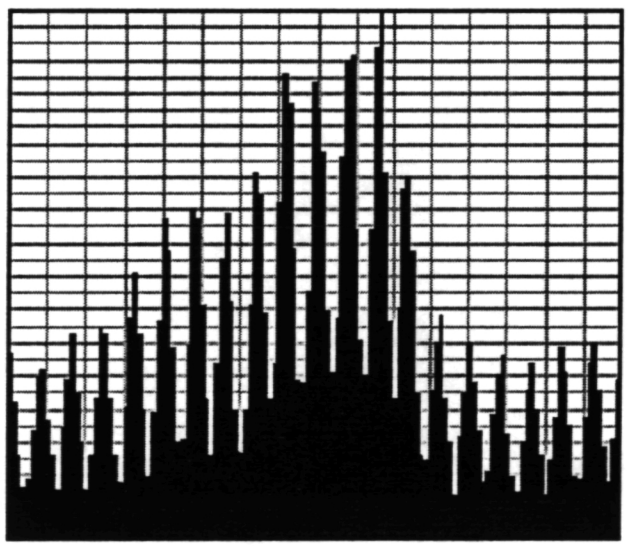


Figure 44. CCD camera noise

It was determined that video signals from the camera are picked up and amplified by the variable gain amplifiers, thereby creating a periodic signal even when the high voltage is off. This source of

error in the data is eliminated by applying power to the camera only during image alignment. Finally, the gain and discriminator thresholds for all 64 channels were readjusted so as to provide the most uniformly matched set of profiles obtainable. Although the above tasks were completed during daylight hours of the first day, bad weather degraded seeing conditions throughout the trip, making it impossible to test the instrument against a star. Consequently, the instrument was stored at the telescope facility in anticipation of the third field test.

3. Third Field Test

During the third field test, Andromeda Gamma was used as the test binary. Andromeda Gamma comprises a primary star with a visual magnitude of 2.12 and a secondary star with a visual magnitude of 5.08. The angular separation of the stars is quoted in Burnham as 10 arcseconds maximum with a 0.5 arcsecond separation occurring in the 1991/1992 timeframe.

Figure 45 represents a consolidation of several profiles of Andromeda Gamma taken over a 15 minute interval and normalized relative to the total number of counts in each profile. With Andromeda Gamma centered on the optical axis of the profiler, we selected a 5x microscope objective and acquired a sequence of data profiles of 1000 scans each. These scans were obtained at a rate of 25 scans per second.

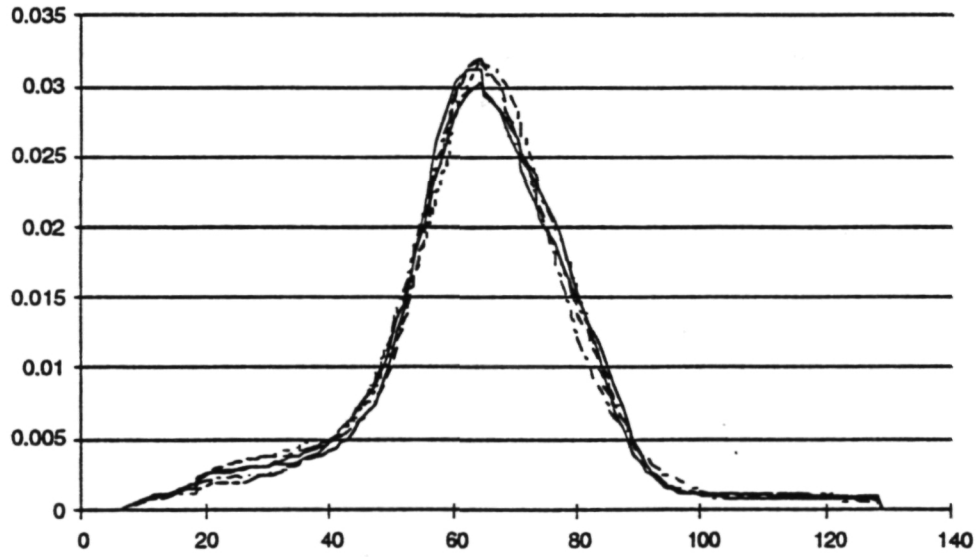


Figure 45. Unprocessed profiles of Andromeda Gamma

The data samples of Andromeda Gamma were interleaved with scans of the apparently singular reference star, Perseus Beta Algol, a magnitude 2.14 star. Figure 46 presents a consolidation of

several profiles of Perseus Beta Algol. As in the Andromeda Gamma data set, these profiles were taken over a 15 minute interval and were normalized to the total number of counts in each profile.

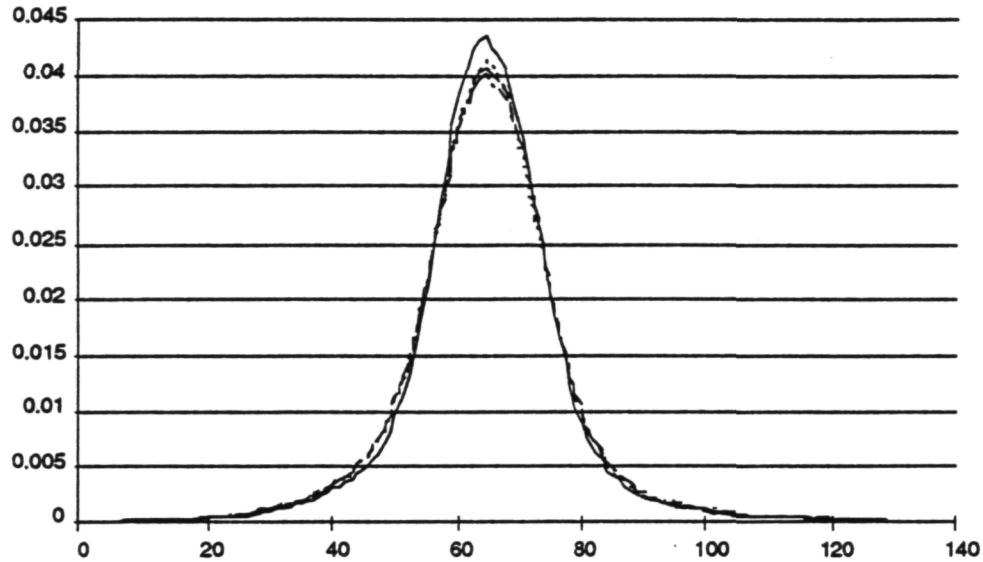


Figure 46. Unprocessed profiles of Perseus Beta Algol

Figure 47 presents a screen shot of the reference star, Perseus Beta Algol. This figure demonstrates the degree to which problems encountered during the first two field trips were corrected prior to the third trip. The image of the previously out of focus star is now a sharp image (compare with Figure 43). The profile also indicates the low number of noise counts in the wings of the profile, together with suppression of the previously encountered profile shoulder phenomenon.

The Strehl intensity ratio feature of the profiler system was used to select superior data sets for further processing. Software code for this purpose (see Appendix F for a full listing) was developed using Symantec's Think C for the Macintosh version 4.0. This software processed all the data files and created a report which provided the file name and the Strehl intensity ratio. Using this report, the top ten percent of the data runs were selected for further processing.

The data analysis software (presented in Appendix H) reads in data from both the test star and a reference star. Each profile is normalized to the total number of counts in the profile. The normalized data sets are Fourier transformed and low-pass filtered to remove noise. The Fourier transformed test profile is then subtracted from the reference profile; the resulting residual data array being inverse Fourier transformed.

The difference profile presented in Figure 48 is typical of processed data obtained in the above manner. In this example, we see the difference profile which results when Andromeda Gamma is analyzed with respect to the reference star, Perseus Beta Algol. Despite the far-from-ideal seeing

condition, it is seen that good correlation exists between four separate data sets. Further, the form of the data is consistent with that predicted for the system (see, for example, Figure 12b). As expected for Andromeda Gamma, the secondary source located to the left of scan center is clearly evident.

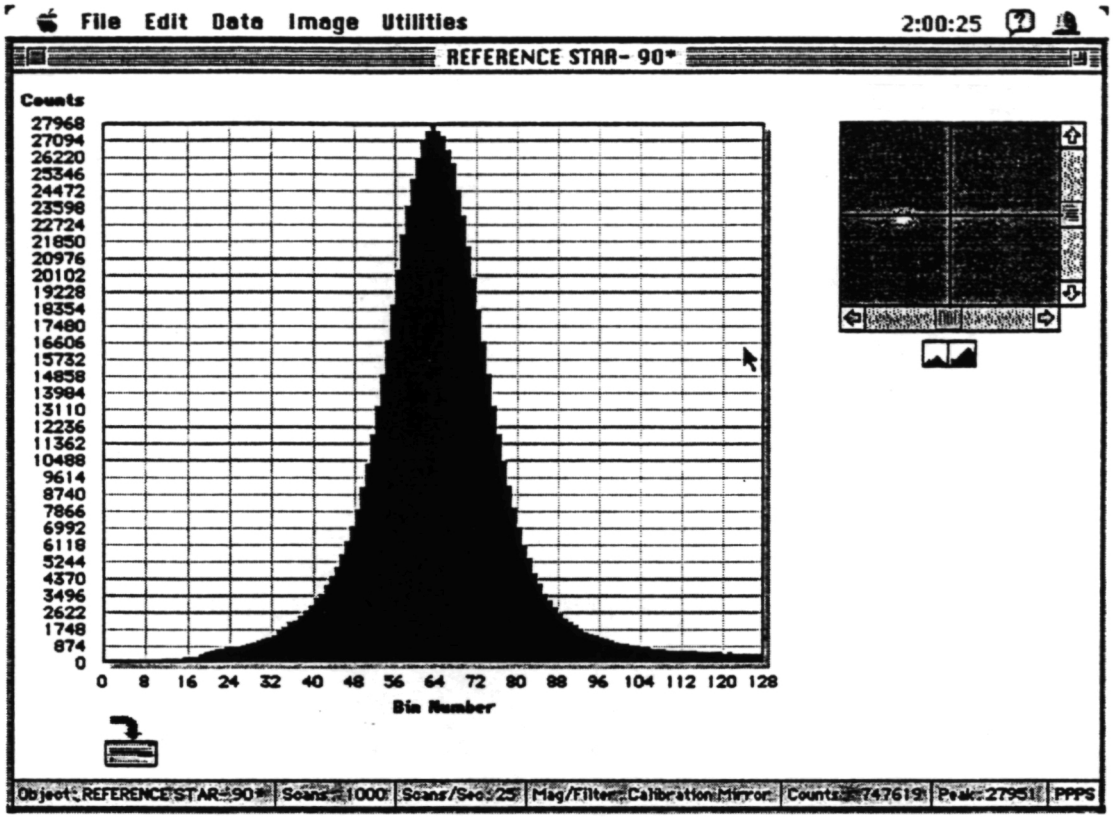


Figure 47. Screen shot of Perseus Beta Algol (reference star)

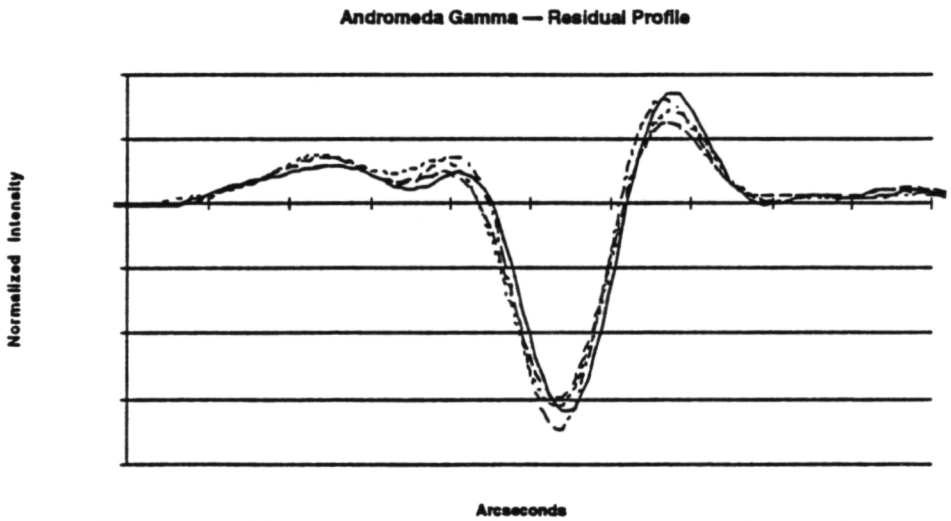


Figure 48. Difference profile for Andromeda Gamma relative to Perseus Beta Algol

THIS PAGE INTENTIONALLY LEFT BLANK

VI. NAME AND PHONE NUMBERS OF PERSONS PREPARING THIS REPORT

Name	Phone Number	Extension
Elena Morris	(619) 755-0700	106
Graham Flint	(619) 755-0700	122
Robert Slavey	(619) 755-0700	169

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: IMAGE PROFILING BY COMPUTER SIMULATION

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A

IMAGE PROFILING BY COMPUTER SIMULATION

Within the scope of the Phase I program, the generation of image profile data via slit scanning of an image occurred toward the end of the program. Thus to examine signal processing techniques at an early stage of the program, it was necessary to develop a computer model for the simulation of such data.

To maximize the extent to which the model can be understood by the casual reader, it has been written in the format utilized by MathCAD[1], rather than in the language which will be employed during Phase II. Consequently, the equations and graphs which are presented in this appendix are identical to those which appear in the program itself. In this regard, it should be noted that MathCAD uses the symbol ":=" to denote "conditionally equal" and requires that a "dot product" be used to denote the multiplication of one function by another. Thus, in MathCAD, a familiar equation such as y = 3x, becomes y:=3·x.

To provide a realistic simulation, the model employs a full Bessel function Airy disc superposition of a primary and a secondary source. The parameters describing the composite source include the intensity, N, of the primary source (given in terms of the number of photoelectrons accumulated within a central bin of a multiple scan); the relative brightness, β, of the secondary source; and the angular separation, θ, of the secondary source with the respect to the primary. The parameters describing the scan system include the width, w, of the scan; the height, h, of the scanning slit; the number, M, of bins into which the profile is subdivided; and the fractional scanning speed error, δf, which occurs relative to the profile of a reference source. To preserve scalability to any optical system, the geometric parameters describing scan width, slit height, and primary/secondary separation are written in units of the diffraction limited angle, λ/D.

For a profile which lies between the scan limits of -w/2 and +w/2, it is convenient to divide the scan into M bins of equal width. When the spatial frequency of the profile is low in comparison to the bin density, we can the describe an ideal (non-noisy) profile in terms of its value at the center of each bin. This allows us to describe the profile in terms of a range variable x(a) which takes the form:

$$x(a) := \left[\frac{2 \cdot a - M - 1}{2 \cdot M} \right] \cdot w \qquad \text{(A-1)}$$

---

1. MathCAD, Version 2.0, MathSoft, Inc., One Kendall Square, Cambridge, Massachusetts 02139.

For the purpose of this analysis, it is assumed that the image profile is rotationally symmetric, and that it has the form of an Airy disc. Further, it is assumed that the Airy disc is produced by narrow band radiation, such that the resulting profile exhibits high-frequency structure. In polar coordinates, the intensity of an Airy disc can be described in terms of the J1 Bessel function and takes the form:

$$I(r) := \left[\frac{2 \cdot J1\ (\pi \cdot r)}{\pi \cdot r} \right]^2 \qquad \text{(A-2)}$$

If we transpose this distribution into rectilinear coordinates and, for each value of x, integrate over y between the limits of -h/2 and +h/2, we obtain the relative intensity transmitted by a narrow slit located at x. Assuming the distribution to be centered at x = 0 (i.e., at the center of the scan), we then can write a profile function, $I_p(x)$, for a primary source in the form:

$$I_p(x) := 2 \cdot \int_0^{\frac{h}{2}} \left[\frac{2 \cdot J1 \left[\pi \left(x^2 + y^2 \right)^{0.5} \right]}{\pi \cdot \left(x^2 + y^2 \right)^{0.5}} \right]^2 dy \qquad \text{(A-3)}$$

Plotting the above function on a logarithmic scale between scan limits of -w/2 and +w/2 produces a profile of the form shown in Figure A-1. For this specific profile, it has been assumed that the scan width is given by $w = 16\,\lambda/D$, where λ is the median wavelength and D is the aperture of the optical telescope.

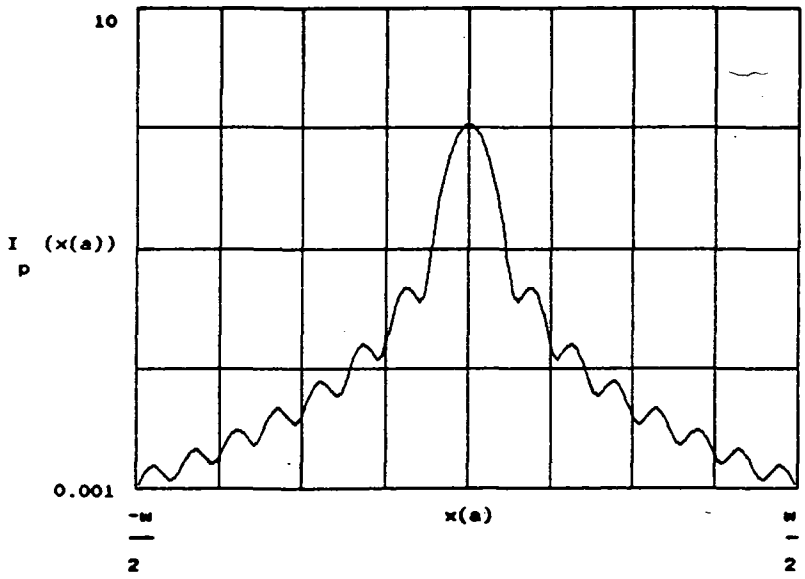


Figure A-1. Slit Scanned Profile of a Single Source.

Typical values for the number of bins and the slit height are 128 and $32\lambda/D$, respectively. Using these values, we find that the value of $I_p(x)$ for each of the central bins becomes 1.0695. When $I_p\{x(a)\}$ is summed over all 128 bins, we obtain a value of 10.002. Thus, the central bin value is sufficiently close to unity that, for statistical purposes, it is not necessary to perform a normalization. The summed value indicates that the effective width of the profile is equivalent to about 10 bins; i.e., to about 1/13 of the scan width.

Introduction of a secondary source is effected by adding to $I_p(x)$ a second function, $I_s(x)$, of the form:

$$I_s(x) := 2\cdot\beta\int_0^{\frac{h}{2}}\left[\frac{2\cdot J1\left[\pi\cdot\left[(x-\theta)^2+y^2\right]^{0.5}\right]}{\pi\cdot\left[(x-\theta)^2+y^2\right]^{0.5}}\right]^2 dy \qquad \text{(A-4)}$$

where β is the relative intensity of the secondary source and θ is its displacement along the x-axis relative to the position of the primary source. The summation creates a combined image profile, $I_{ps}(x)$. For most cases of interest, the value of β is very small (typically in the range $10^{-5} < \beta < 10^{-3}$). Consequently, when the primary and secondary distributions are summed and plotted in the manner of Figure A-1, the presence of the secondary source is visually undetectable.

In a real image profiler, the centroid of a combined image is centered with respect to the discrete bins which comprise a scan. Thus, in the case of 128-bin scan, the sum of the counts in bins 1 through 64 is equal to the sum of the counts in bins 65 through 128. However the distribution described by $I_{ps}(x)$ does not satisfy this condition, insofar as the primary function is centered, while the secondary function is displaced from the center. To correct this imbalance, it is necessary to introduce a small offset, δ, of the combined function with respect to the center of the scan. In the presence of such an offset, a function similar to $I_{ps}(x)$ can be described by two integrals; one representing the portion of the profile which lies to the left of the scan center, and one representing the portion to the right. If we denote these integrals by $I_{psl}(\delta)$ and $I_{psr}(\delta)$, respectively, then we can write:

$$I_{psl}(\delta) := \int_{\frac{-w}{2}+\delta}^{\delta} I_{ps}(x)\,dx \qquad \text{(A-5)}$$

and

$$I_{psr}(\delta) := \int_{\delta}^{\frac{w}{2}+\delta} I_{ps}(x)\,dx \qquad \text{(A-6)}$$

In the MathCAD format, the value of δ is obtained by solving a root equation of the form:

$$\delta\theta := \text{root}\left[I_{psl}(\delta) - I_{psr}(\delta), \delta\right] \tag{A-7}$$

Solution of the equation is effected by finding the value of δ for which $I_{psl}(\delta) - I_{psr}(\delta)$ is zero. This value is then returned as $\delta\theta$, the offset required to balance the profile with respect to the scan center.

Having established the value of $\delta\theta$ for a specific set of initial conditions, we can express the centered sum of primary and secondary distributions in terms of the individual components, $I_{p\delta}(x)$ and $I_{s\delta}(x)$; these expressions being:

$$I_{p\delta}(x) := 2\cdot\int_0^{\frac{h}{2}}\left[\frac{2\cdot J1\left[\pi\cdot\left[(x-\delta\theta)^2+y^2\right]^{0.5}\right]}{\pi\cdot\left[(x-\delta\theta)^2+y^2\right]^{0.5}}\right]^2 dy \tag{A-8}$$

and

$$I_{s\delta}(x) := 2\cdot\beta\int_0^{\frac{h}{2}}\left[\frac{2\cdot J1\left[\pi\cdot\left[(x-\theta+\delta\theta)^2+y^2\right]^{0.5}\right]}{\pi\cdot\left[(x-\theta+\delta\theta)^2+y^2\right]^{0.5}}\right]^2 dy \tag{A-9}$$

In a real profiling system, we must assume small fluctuations and drift in the angular rate of the scanning process. In the context of a system which performs a set of scans relative to a singular reference source, followed by a similar set relative to a composite, or unknown, source, the resulting profile errors are defined by the difference in the mean scan rates associated with the two sets. Thus, to simulate a real system, we can introduce a fractional scan rate error into the centered reference profile. If the fractional rate error is defined as δf, the reference profile then takes the form:

$$I_{ref}(x) := 2\cdot\int_0^{\frac{h}{2}}\left[\frac{2\cdot J1\left[\pi\cdot\left[(1+\delta f)^2 x^2+y^2\right]^{0.5}\right]}{\pi\cdot\left[(1+\delta f)^2 x^2+y^2\right]^{0.5}}\right]^2 dy \tag{A-10}$$

The data processing employed in a real system performs a normalization of the bin counts so that the sum over all bins for the reference source is equal to a similar sum for the unknown source. This establishes the amplitude, β_{ref}, for the reference profile which subsequently is subtracted from

the unknown profile. In the context of computer simulation, the value of β_{ref} is given by:

$$\beta_{ref} := \frac{\int_{\frac{-w}{2}}^{\frac{w}{2}} \left[I_{p\delta}(x) + I_{s\delta}(x) \right]}{\int_{\frac{-w}{2}}^{\frac{w}{2}} I_{ref}(x)\, dx} \qquad \text{(A-11)}$$

It is assumed that the photoelectron detection process is a random process. Consequently, the number of photoelectron counts within each bin of a profile is expected to exhibit a probability distribution which is normal; i.e., to exhibit a noise level wherein the *one-sigma* value for a specific bin is equal to the square root of the mean number of counts in the bin. A convenient means for introducing such a noise to each bin of the distribution involves the use of two random numbers in a root equation. First, we define the random numbers; s_a being a random number between zero and one, and r_a being a random number between zero and two. Second, we define the mean number, N, of photoelectron counts associated with the most densely populated bins at the center of the distribution. This allows us to define the mean number of counts within any bin of a combined profile as $N \cdot I_{ps\delta}[x(a)]$. A normally distributed error, ϵ_a, then can be obtained for each bin via the root equation:

$$\epsilon_a := \text{root}\left[s_a - \text{erf}\left[\frac{B}{\left[2 \cdot N \cdot I_{ps\delta}\ [x(a)] \right]^{0.5}} \right], B \right] \qquad \text{(A-12)}$$

The above expression returns a number for ϵ_a which corresponds to a randomly selected number between zero and one of the cumulative error function; the *one-sigma* value of the error being equal to the square root of the mean count, $N \cdot I_{ps\delta}[x(a)]$. Having established an amplitude for each error, random signs are then introduced by expressions of the form $[(1-r_a)/|(1-r_a)|]$. This allows us to write an expression for a noisy combined source profile in the form:

$$I_{ps\delta n}(a) := N \cdot I_{ps\delta}\ [x(a)] + \left[\frac{1 - r_a}{|1 - r_a|} \right] \cdot \epsilon_a \qquad \text{(A-13)}$$

Typically, the number of scans relative to a reference source is significantly larger than that relative to a composite, or unknown, source. Consequently, the noise associated with the reference profile usually can be neglected. (In the case of similar cumulative counts in reference and unknown profiles, the noise term in Equation (A-13) must by multiplied by $\sqrt{2}$.) Thus,

the bin-by-bin difference between the unknown profile and the reference profile becomes:

$$\delta I_{ps\delta n}(a) := I_{ps\delta n}(a) - N \cdot \beta_{ref} \cdot I_{ref}[x(a)] \qquad \text{(A-14)}$$

To illustrate the characteristics of the program, it is convenient to examine a typical example. The difference profile shown in Figure A-2 corresponds to the specific input conditions listed below:

Scan width	(w)	$16\,\lambda/D$
Slit height	(h)	$32\,\lambda/D$
Number of bins	(M)	128
Fractional scan error	(δf)	0.0001
Maximum bin count	(N)	10^8
Secondary intensity	(β)	0.0001
Secondary displacement	(θ)	$4\lambda/D$

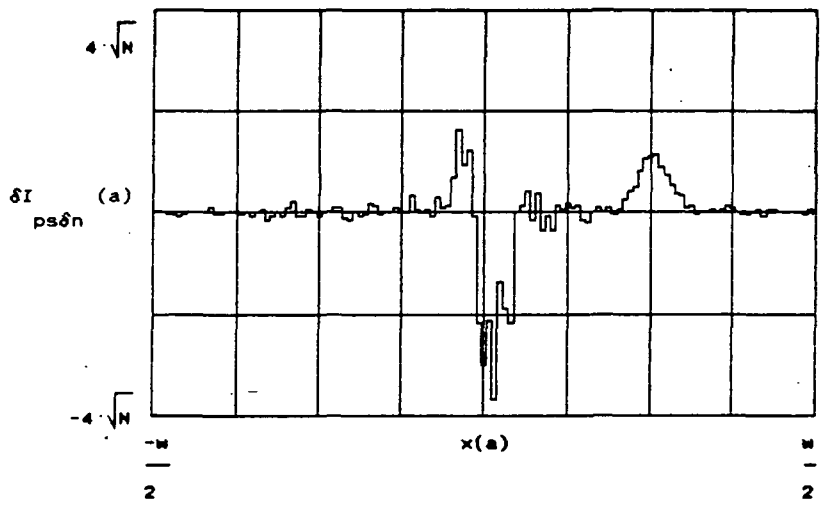


Figure A-2. Unprocessed Noisy Difference Profile.

Examination of Figure A-2 reveals several significant features. First, the presence of a secondary source having an amplitude of about $\sqrt{N}$ and a displacement of about $4\,\lambda/D$ is clearly evident. It should be noted that the vertical scale of the plot extends from $-4\sqrt{N}$ to $+4\sqrt{N}$; i.e., between the limits of ± 4-sigma relative to the largest bin counts. Thus, in terms of primary source intensity, the scale extends from -0.0004 to +0.0004.

The feature at the center of the scan is somewhat more complex; showing an increased noise level, together with a strong minimum to the right of center and a subsidiary maximum to the left of center. The increased noise level reflects the larger photoelectron counts associated with the central bins of the profile. The strong minimum occurs as a consequence of normalizing the reference profile so as to have a total count equal to the sum of the counts encompassed by the unknown profile. As a consequence, in the vicinity of the scan center, the normalized intensity of the reference is greater than that of the primary contribution to the unknown profile. Thus, when the

reference profile is subtracted from the unknown profile, a minimum occurs in the vicinity of the scan center. However, because the profile of the primary is offset from center due to the presence of a secondary, it is displaced slightly to the left of the reference. Thus, to the right of center, the less intense primary is weaker than the reference by an amount which is greater than the intensity of the secondary. Meanwhile, to the left of center, the displaced primary is slightly more intense than the reference. In the absence of noise, the areas which represent the secondary source and the arithmetic sum of the central maximum/minimum are equal and opposite.

In light of the above noted identity, the presence of a secondary source can, in principle, be deduced either directly from the presence of a secondary profile, or indirectly from the asymmetry of the central maximum/minimum. However, analysis of the central maximum/minimum yields only the approximate magnitude and the direction of secondary displacement. Also, the region of the central maximum/minimum is intrinsically noisier than the region of the scan which encompasses the displaced secondary. Thus, in practice, the most efficient signal processing algorithms are based upon analysis of the secondary profile, rather than upon analysis of the central maximum/minimum.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: FAST FOURIER TRANSFORM FILTERING

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B

FAST FOURIER TRANSFORM FILTERING

The Phase I program has produced a breadboard image profiling system which generates a profile in the form of discrete bin counts. The noise associated with each count is consistent with a normal distribution wherein the *one-sigma* noise amplitude for a bin is equal to the square root of the mean number of counts in the bin. If the counts within adjacent bins were uncorrelated, this noise level would establish the limit of sensitivity for the system. However, since the profile of an unresolved image encompasses several bins, there exists a degree of correlation between neighboring bin counts.

The signal information associated with interbin correlation can be used to improve the final signal-to-noise ratio by a variety of spatial filtering, and/or profile smoothing techniques. Within the scope of Phase I, several such techniques have been examined; the most successful being that of fast Fourier transform filtering.

For the purpose of demonstrating the fast Fourier approach, both computer generated and optical scanner generated profiles have been processed via the MathCAD fast Fourier transform routines. The *fft* function employed by MathCAD returns the fast Fourier transform of a vector of real data representing measurements at regular intervals in the time domain. The result is a vector of complex coefficients representing values in the frequency domain. For a fast Fourier transform of real data, the second half of the transform is the conjugate of the first. MathCAD takes advantage of this characteristic by discarding the second half of the result vector; returning a vector only half as large as the argument vector, plus one.

The *ifft* function employed by MathCAD return the inverse Fourier transform of a vector of data representing values in the frequency domain. The result is a vector representing values in the time domain. The *ifft* function is the inverse of the *fft* function applied to read data in the time domain. To compute a result, MathCAD first creates a new vector by taking the conjugates of the *fft* elements and appending them to the original elements. The result is a vector containing a number of elements which is two less than twice the argument. Thus, for the purpose of filtering real data in the time domain, the *fft* and *ifft* functions are exact inverses.

To filter a digitized image profile, such as that presented in Figure A-2 of Appendix A, it is necessary to define a new range variable, *i*, such that:

$$i := 0 \,..(M - 1) \tag{B-1}$$

$$x_i := \left[\frac{2 \cdot i - M + 1}{2 \cdot M}\right] \cdot W \tag{B-2}$$

Using this definition, we can derive a fast Fourier transform of a noiseless image profile where the elements, A_j, in the frequency domain are defined in terms of the range variable, j. For an intensity profile containing 128 bins, the number of elements in the transform is 65. However, in the case of a noiseless, Airy disc profile contained within a scan width of $w\lambda/D$, only the first w elements are non-zero. The manner in which the an amplitude decreases with frequency can be seen from an examination of Figure B-1.

On the other hand, when a noisy profile is transformed, all 65 elements are found to be non-zero. The transform presented in Figure B-2 corresponds to the noisy profile shown as Figure A-2 of Appendix A. First-order filtering of such a profile can be performed by setting all but the first w elements of the transform to zero, and deriving the inverse transform; these results are shown in Figure B-3. As can be seen, the first-order filtering process provides a significant amount of signal smoothing. In particular, the magnitudes and locations of the signal and maximum/minimum features described in Appendix A are greatly enhanced.

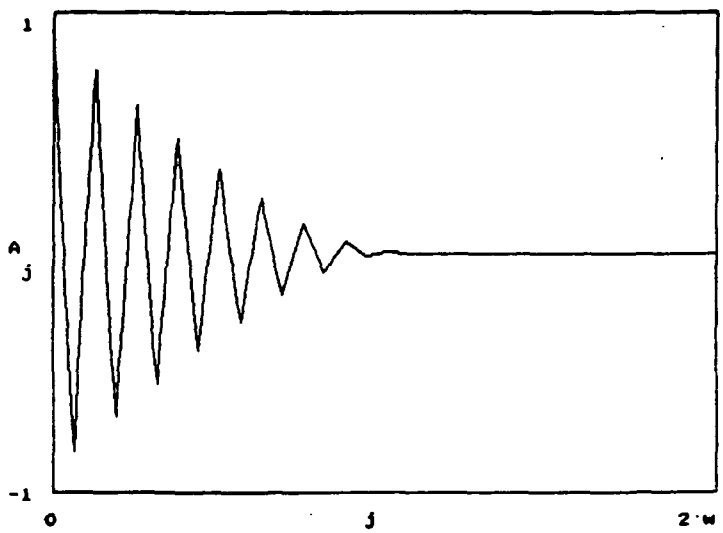


Figure B-1. Fourier Transform of a Noiseless Reference Profile.

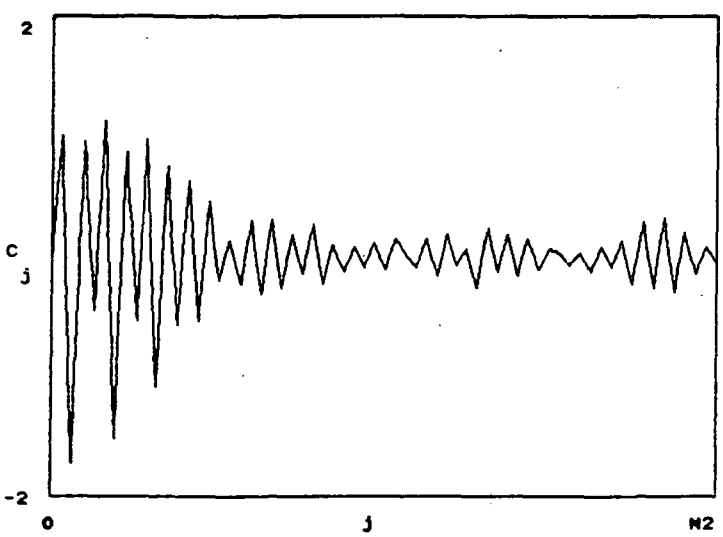


Figure B-2. Fourier Transform of a Noisy Composite Profile.

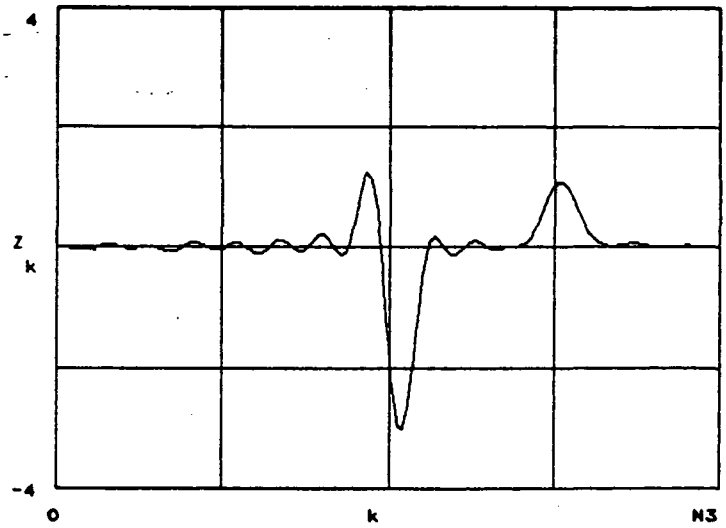


Figure B-3. Filtered Inverse Fourier Transform of a Noisy Composite Profile.

A second-order filtering process involves a correction of the reference signal to account for the presence and location of the suspected secondary source. For this purpose, we have developed a profile search algorithm which examines the first-order profile and identifies the approximate amplitude and location of a secondary source. The values so derived are used to revise the amplitude and position of the normalized reference profile. The revised profile is then subtracted from the original noisy profile. The result is refiltered by the fast Fourier transform process. In the case of the profile shown in Figure B-3, this second filtering process yields the profile shown in Figure B-4.

The residual maximum/minimum/maximum feature at the center of the profile is associated with the fractional scan error that was introduced into the original reference profile. In this case, the reference scans were run at

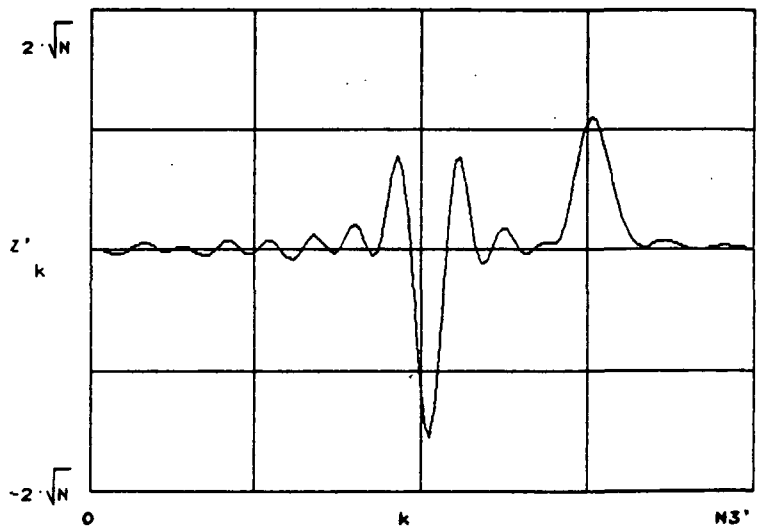


Figure B-4. Computer Recentered and Filtered Profile. (Note that the vertical scale has been enlarged by a factor of two.)

an average angular rate which was high by a factor of 0.0001. Had the average rate been slow, the phase of the feature would be reversed; i.e., it would appear as a minimum/maximum/minimum.

Since the second order filtering process corrected the amplitude of the reference, the sum of the areas encompassed by the two maxima are approximately equal to the area encompassed by the minimum (see Fig. B-4). Using this approximate identity, allows a second search algorithm to identify the amplitude and sign of the scan error. This, in turn, provides a second order correction to the reference profile. Subtracting the more precisely corrected reference from the noisy profile, and subjecting the result to a third iteration of fast Fourier transform filtering, yields the final profile shown in Figure B-5.

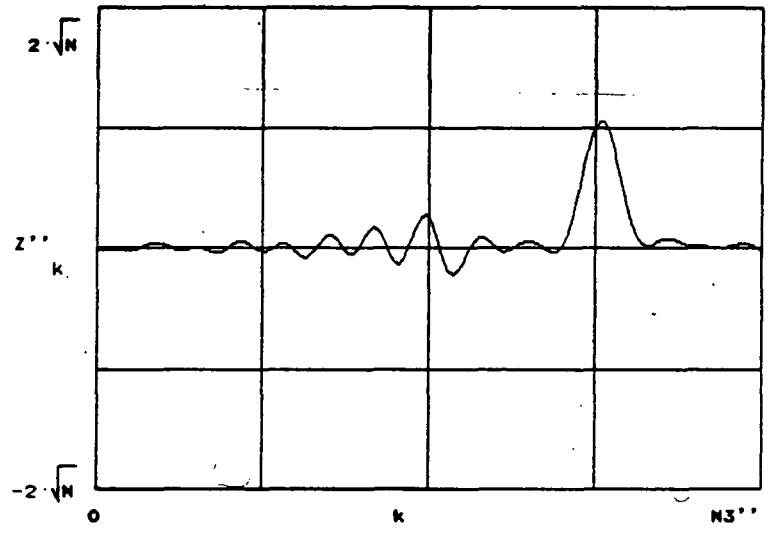


Figure B-5. Fully Corrected and Filtered Profile.

C-2.

APPENDIX C: PRESCALER STATISTICS

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

PRESCALER STATISTICS

In prior discussions of signal processing, it has been assumed that an incoming photoelectron stream is converted into a pulse sequence which is clocked directly from the preamplifier into a shift register. This also has been the approach taken experimentally throughout Phase I. However, during Phase I, it became increasingly apparent that the introduction of a prescaler between the preamplifier and the shift register could offer advantages both in performance and cost. (A prescaler performs pulse rate division by forwarding a single pulse to the shift register at the conclusion of each period within which a specified number of pulses are received from the preamplifier.) Thus, within the scope of our analytic studies, we have developed a computer model of a pulse processing system which includes a prescaler.

The advantages of a prescaler are threefold. First, it provides an upward expansion of the dynamic range of the profiling system. Second, it reduces the electronic bandwidth requirements for most elements of the signal processor. Third, and perhaps most importantly, it greatly reduces the required capacity of the shift registers; thereby effecting a significant cost and size reduction in multichannel systems.

The expansion in dynamic range is best understood via an examination of the dead time coefficient which is intrinsic to any system that counts random events. When the time of occurrence for a sequence of events is random, the probability that an event will occur within any specified interval becomes the product of the interval duration and the mean rate of occurrence. Thus, no matter how short an interval is chosen, the probability that an event will occur within the interval is finite. From an event counting standpoint, any real system exhibits a time interval within which it is incapable of distinguishing between one event and two events. This interval is characteristic of the specific system and is referred to as the system *dead time*. For situations wherein the mean time between events is large compared to the dead time, the fraction of counts which fail to be registered is small. Consequently, it is a simple matter to apply a small correction factor based upon the measured value of system dead time. However, as the mean time between events is reduced and the magnitude of the correction increases, the accuracy with which the real count can be determined becomes degraded. In the context of an image profiling system, this degradation in accuracy sets an upper bound to the count rate which can be accommodated; i.e., it establishes the upper limit of the system dynamic range.

For a given system, the relationship between the true count rate, N, and the observed count rate, n, can be expressed by:

$$n = Ne^{-N\tau_d}, \qquad \text{(C-1)}$$

where the dead time coefficient, τ_d, is defined in terms of the value of N for which the observed rate falls to 1/e of the true rate. Unfortunately, the above expression does not provide a simple arithmetic means for deriving the true count rate from the observed rate. Consequently, in the context of high speed image profiling, the appropriate count correction is best applied via extrapolation from a look-up table contained within the computer.

In the absence of a prescaler, the dead time of an image profiler can be no less than the period between shift register clock pulses. Throughout the Phase I program, we have used a clock rate of 20 megahertz; yielding a theoretical dead time of 50 nanoseconds. In practice, however, the presence of waveform imperfections and logic pulse jitter, cause the effective dead time to be somewhat longer; usually on the order of 60 nanoseconds. The maximum practical count rate consistent with this dead time is by no means clear cut. In particular, it depends strongly upon the accuracy which must be achieved and upon the effort one is prepared to expend in calibrating the system. However, for the purpose of both the analytic and the experimental activities of Phase I, it has been assumed that a maximum value of 10^5 counts per second (c/s) is desirable, while 10^6 c/s represents an upper limit.

It is instructive to examine the above limits in the context of an image profiling system located at the focal plane of the Hubble Space Telescope. For this purpose, we assume a profiling system which employs a 150-millimicron bandwidth centered at 425 millimicrons and which exhibits an angular resolution equivalent to one-tenth of the resolution of the telescope. Also, we assume that the shortest possible measurement time is desired; i.e., that incoming signal intensity must not be sacrificed by the insertion of an optical attenuator. Under these circumstances, the visual magnitude of a source corresponding to the 10^6 c/s upper limit is approximately 6.5. For the more desirable rate of 10^5 c/s, the brightest source is reduced to visual magnitude 9. In either case, it is evident that the intensity of many sources of interest would exceed the system dynamic range by a significant factor.

Within the present state of the art, it is impractical to increase dynamic range significantly by increasing the clock rate of the shift registers. Also, recognizing that an increase in clock rate must be matched by an increase in shift register capacity, such an approach would be prohibitive from a cost standpoint. On the other hand, prescalers with bandwidths of several hundred megahertz are readily available. A question which must be addressed, therefore, is "Does the use of a prescaler impair the statistical precision of an image profiler?"

In the absence of a prescaler, the profile derived by each channel from a single scan is centered electronically to within one photoelectron count. However, the true accuracy of such a centration process is set by the statistical uncertainty associated with the magnitude of the total count encompassed by the scan. In a one-megabit shift register, for example, the maximum total count is about 10,000. Thus, in performing an image centration, the system is comparing two numbers with values in the vicinity of 5,000. The *one-sigma* value for each of these numbers is $\sqrt{5000}$, or about 70 counts. If the probable errors are added on an RMS basis, then the *one-sigma* centration error becomes ± 0.01 of the width of the image. Since the image width typically encompasses about one-tenth of the scan, the corresponding scan centration error is about 0.001.

If we employ a prescaler which divides the incoming count by twenty, the maximum uncertainty with which the incoming count is centered within the shift register becomes ± 10 counts. This, is turn is equivalent to a scan centration error of 0.0001; i.e., to an error which is smaller than that intrinsic to the statistics of the original count by a factor of ten. Thus, for the count rates typical of an image profiler, the use of a prescaler with a divide ratio of twenty or less introduces a negligible reduction in image centration accuracy.

Having established that a prescaler can provide a substantial increase in dynamic range without an attendant degradation in accuracy, it becomes appropriate to examine the influence of a prescaler upon shift register requirements. For this purpose, let us assume that the dead time of the preamplifier/prescaler is very short compared to the interval between shift register clock pulses. Let us also assume that the divide ratio of the prescaler is F, and that its count is set to zero at a time t = 0. If the mean rate of arrival of photoelectrons is N sec^{-1}, then the probability that the first photoelectron will arrive within a short time interval defined by $0 < t < \tau$ (assuming $\tau \ll 1/N$) is simply $N\tau$. Likewise, the probability that the first photoelectron will arrive during a subsequent time interval defined by $(n - 1)\,\tau < t < n\tau$ is given by:

$$P_{1,n} = (1 - N\tau)^{n-1} \cdot N\tau. \qquad \text{(C-2)}$$

Extending this argument to the arrival of the F^{th} photoelectron, we find that the probability for the prescaler to reach a count of F during the interval defined by $(n - 1)\,\tau < t < n\tau$ is given by:

$$P_{F,n} = \frac{(1 - N\tau)^{n-1} \cdot (N\tau)^{F} \cdot (n - 1)!}{(F - 1)!\,(n - F)!} \qquad \text{(C-3)}$$

Consistent with our previous discussion, we can assume a value for N of 10^6 counts per second. For a preamplifier/prescaler bandwidth of a few megahertz, an appropriate value of τ becomes 10^{-8} second. If we further assume a prescaler divide ratio, F, equal to 10, the relationship between P_n and n is that shown in Figure C-1. As expected, the peak of the probability curve occurs in the vicinity of n = 1000; i.e., approximately 10 microseconds after the prescaler has been set to zero. However, of greater significance is the extremely low probability that the prescaler will time-out in a period of less than 2 microseconds. The magnitude of this latter probability is shown as a cumulative probability in Figure C-2. Here, we see that the likelihood for the prescaler to time-out within 2 microseconds is less than 10^{-4}.

The above conclusion is of major significance to the sizing of image profiler shift registers. Specifically, the inclusion of a divide-by-ten prescaler allows us to reduce the shift register clock speed from 20 megahertz to 500 kilohertz with no loss in performance. At first glance, it seems surprising that a ten-to-one reduction in count rate allows a forty-to-one reduction in clock speed. However, it must be recognized that the output pulse train from the prescaler is far from random. As shown by Figure C-1, it exhibits a mean interpulse spacing of 10 microseconds together with a *one-sigma* of about 3.5 microseconds.

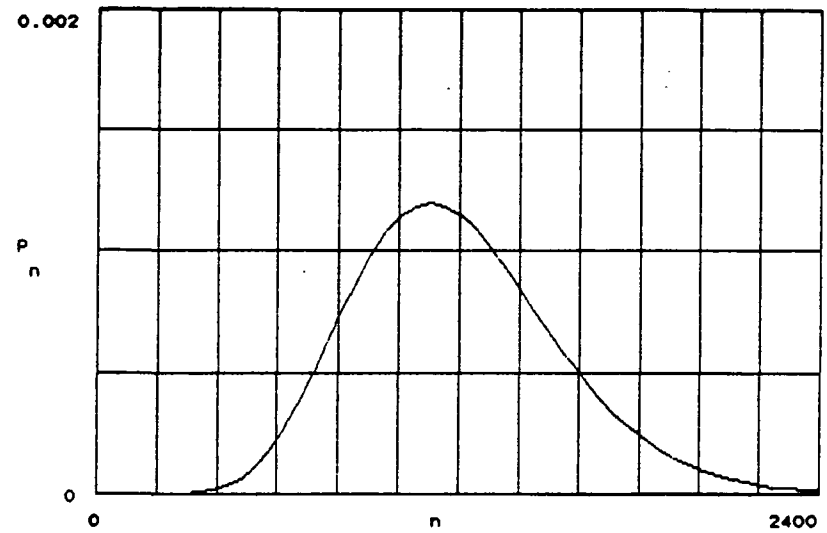


Figure C-1. Interpulse Interval Probability Distribution at the Output of a Prescaler.

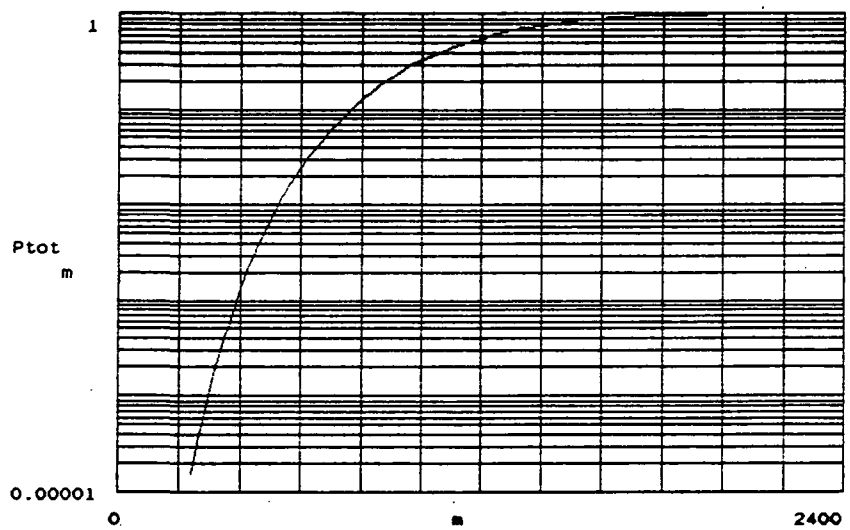


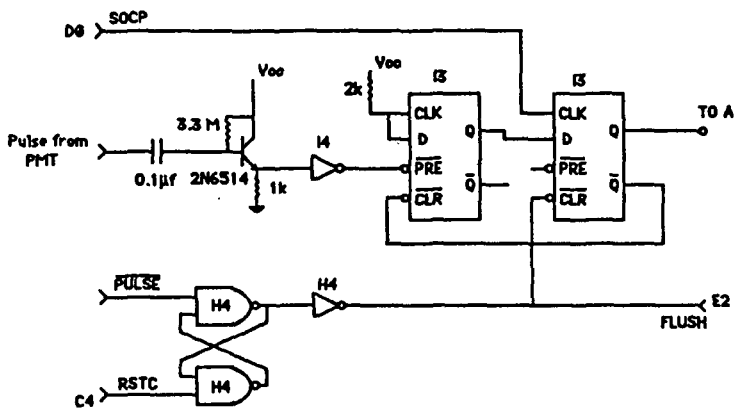
Figure C-2. Cumulative Pulse Arrival Time Probability at the Output of a Prescaler.

From the standpoint of shift register cost, an optimum number of bits in the register probably is 64,000; this being consistent with a scan period of 0.1 second and a clock rate of 640 kilohertz. This value is slightly higher than the 500-kilohertz minimum required by a divide-by-ten prescaler, and yields a probability of about 10^{-5} for losing pulses at the shift register input.

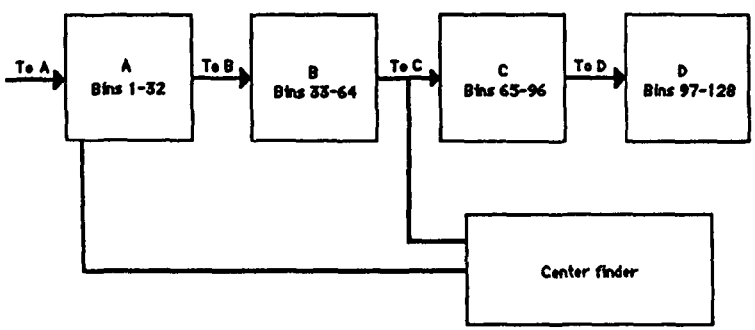
APPENDIX D: SINGLE CHANNEL SIGNAL PROCESSOR SCHEMATICS

THIS PAGE INTENTIONALLY LEFT BLANK

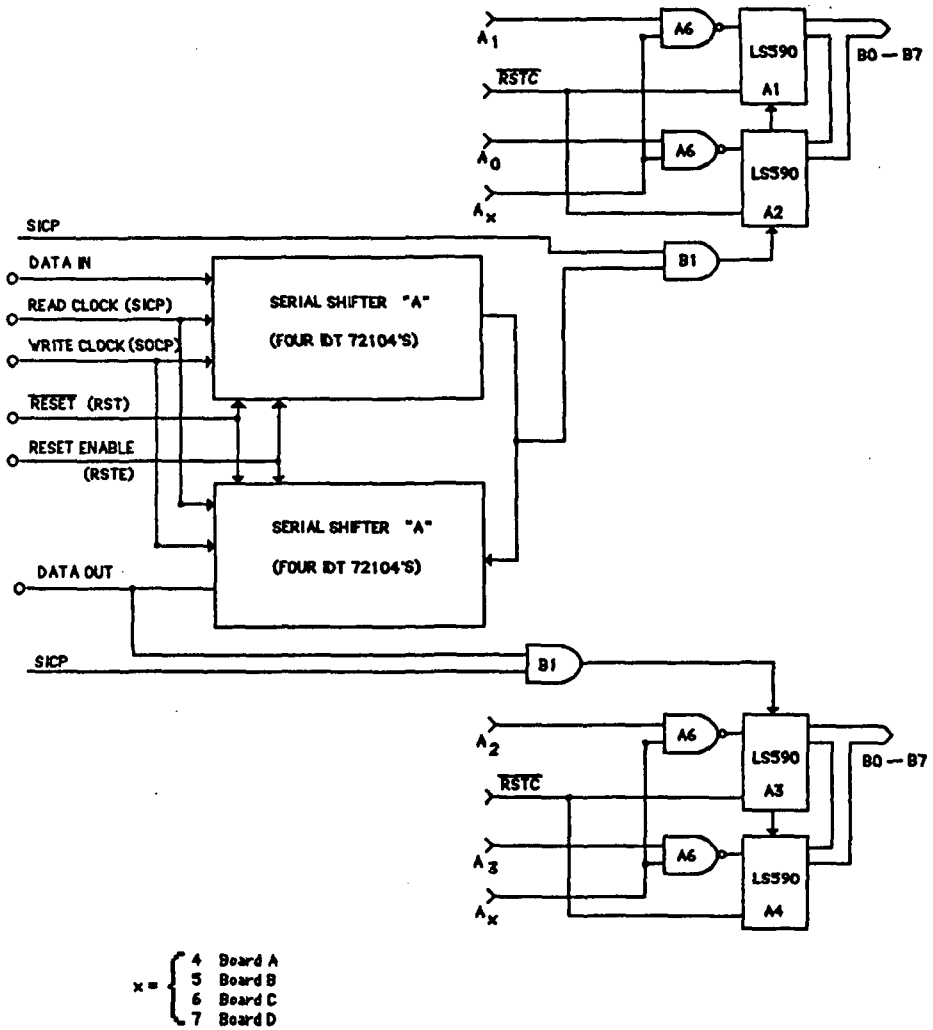
Pulse Detector/Shaper



Center Finder Design

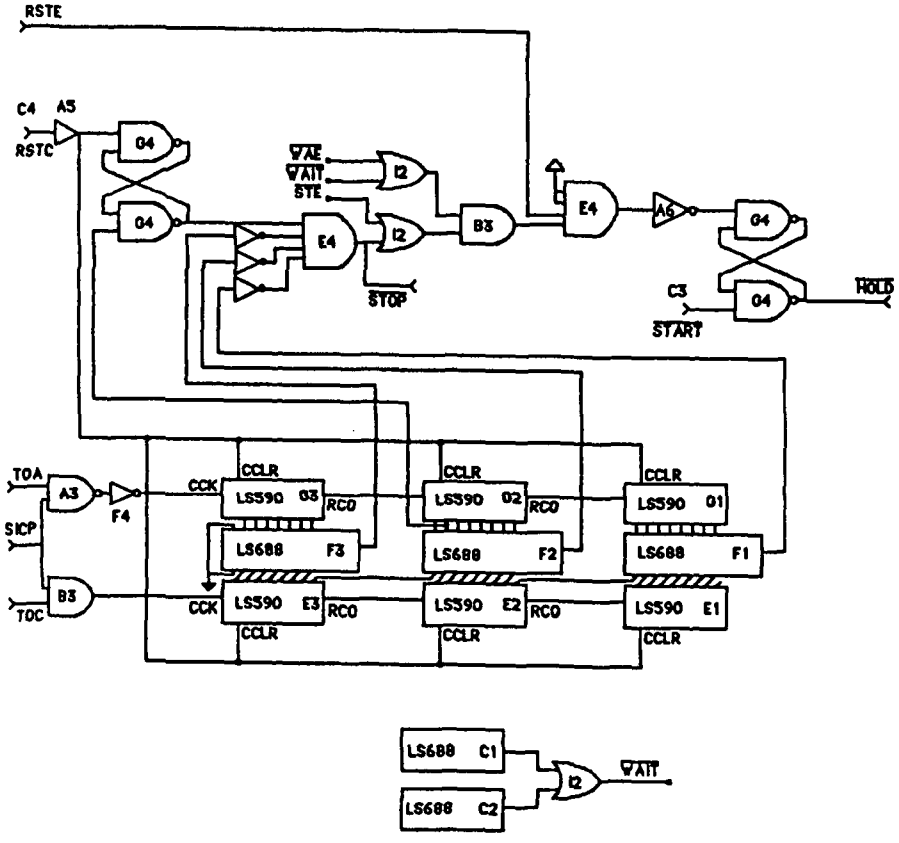


Serial Module Design



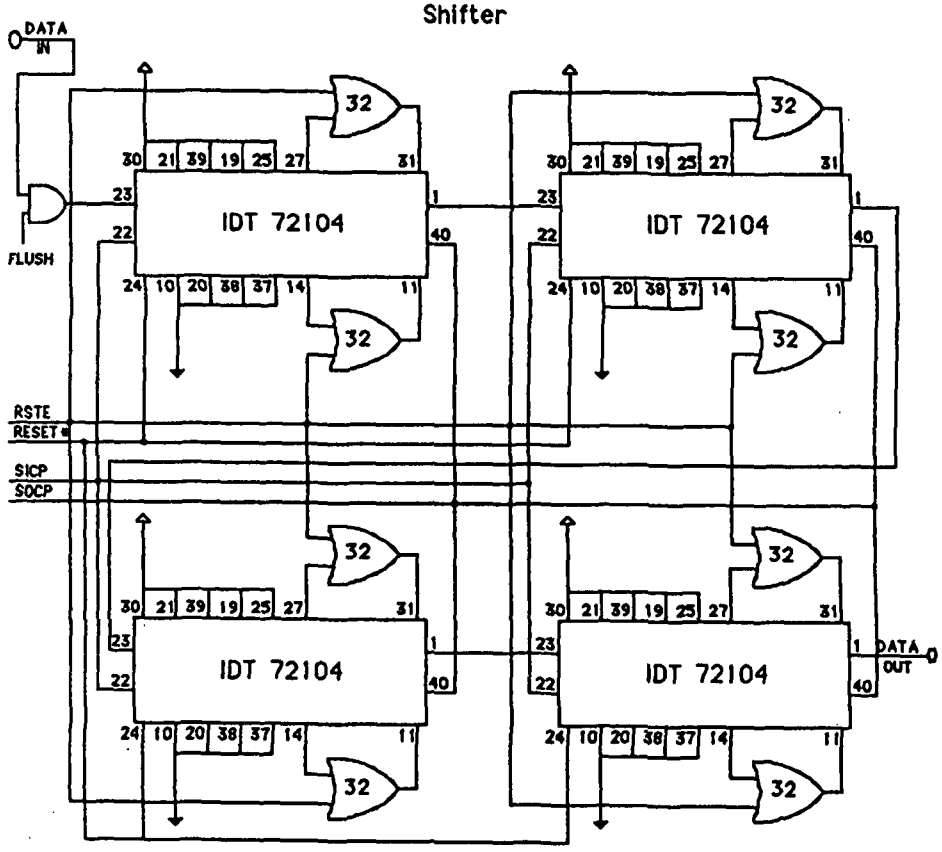
x = { 4 Board A; 5 Board B; 6 Board C; 7 Board D }

Hold Logic / Center Finder

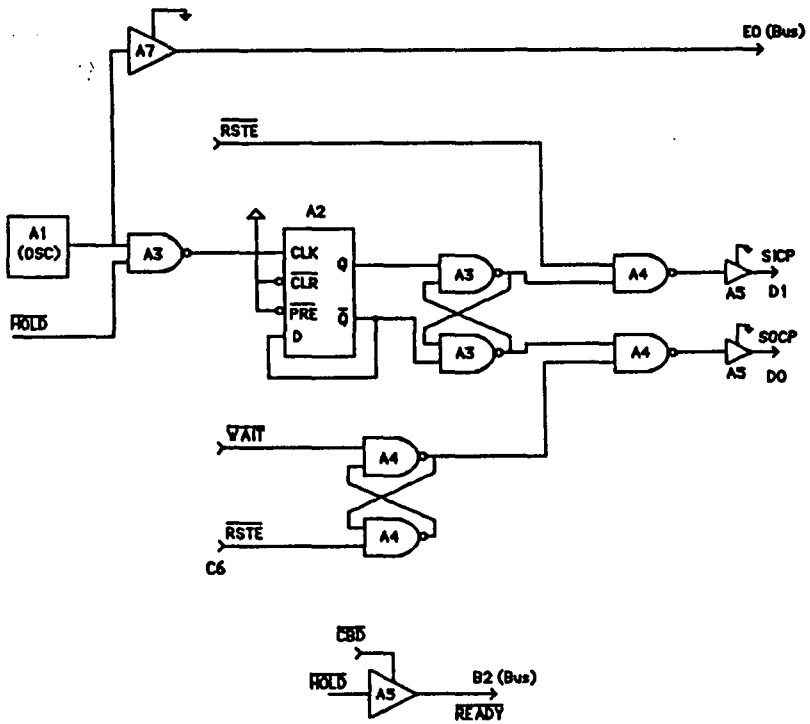


C1 = C2 = 0 when clock has run to latched # of cycles (ie setting SICP/SOCP phase and reading bins)

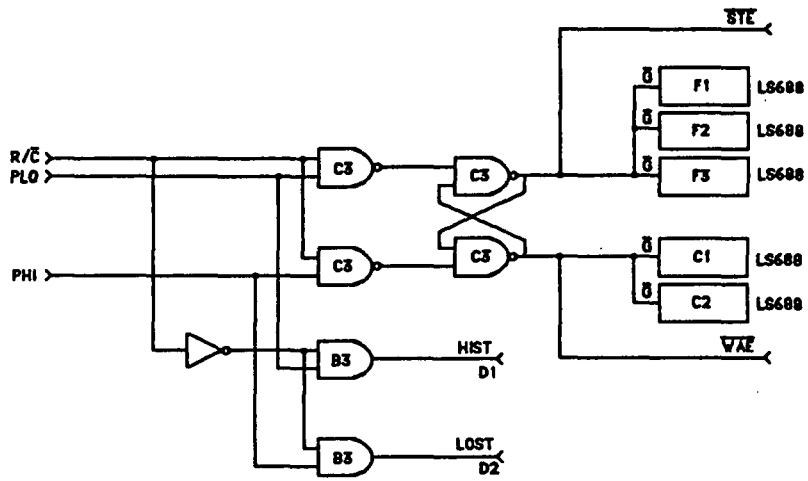
Shifter



Clock Logic / READY



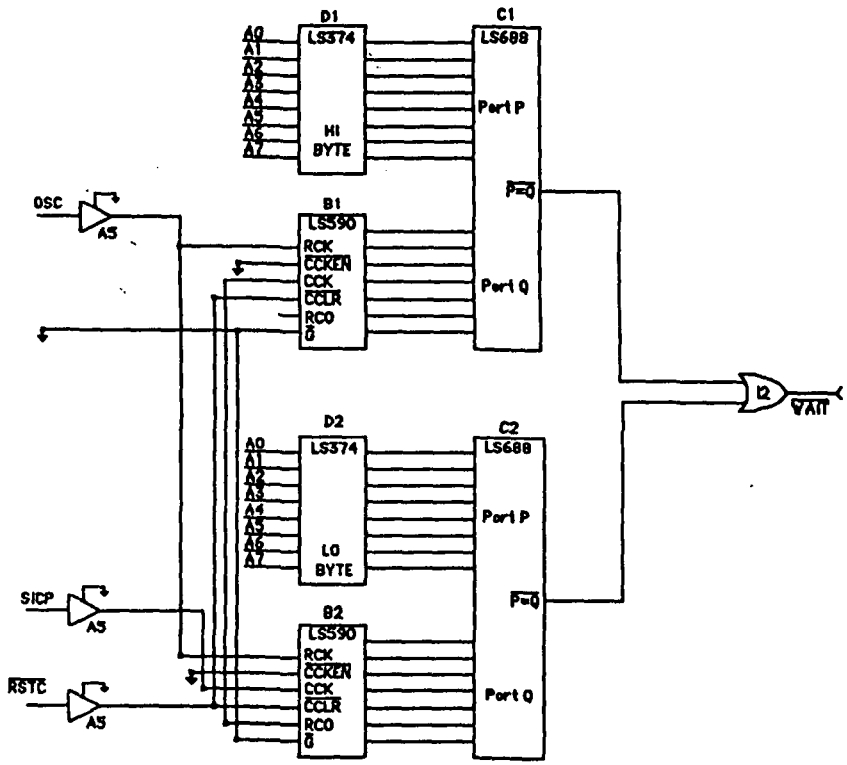
Stop/Wait Select



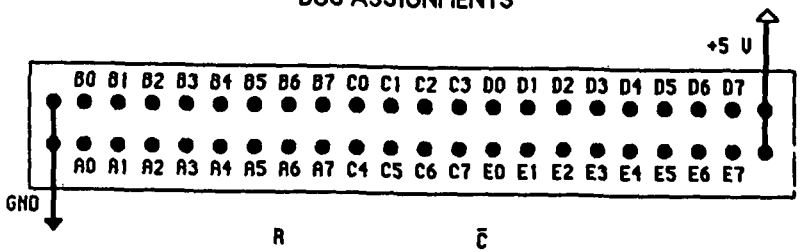
STE - GOES LOW FOR PROFILE CENTERING MODE

WAE - GOES LOW FOR CLOCK CYCLE COUNT MODE

Count-Out Decoding ($\overline{WAIT}$)

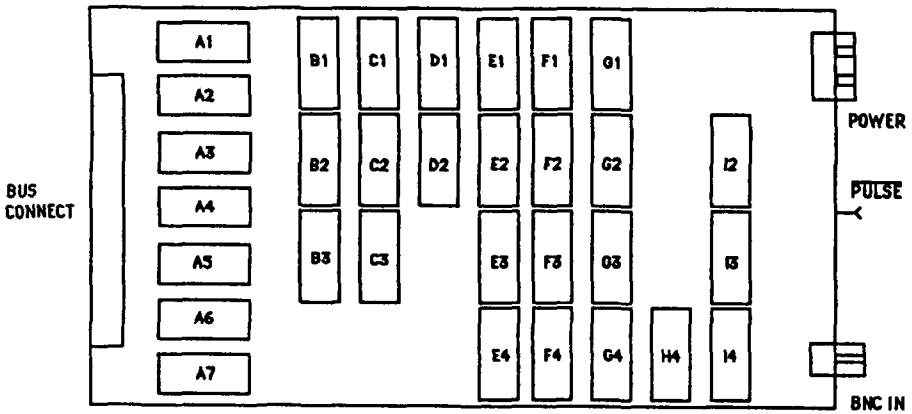


BUS ASSIGNMENTS



	R	$\overline{C}$
A0	SEL0	
A1	SEL0	
A2	SEL2	
A3	SEL3	PRESET TO CLOCK COUNTER (D1 & D2)
A4	SELA	
A5	SELB	
A6	SELC	
A7	SELD	
B0		
B1		
B2		READY
B3		BIT COUNTS TO COMPUTER
B4		
B5		
B6		
B7		
C0	R/$\overline{C}$	
C1	PL0	
C2	PHI	
C3	START	
C4	RSTC	
C6	RSTE	
C7	RST	
D0		SOCP
D1		SICP
D2		TOA
D3		TOB
D4		TOC
D5		TOD
E0	OSC - For Counter Clear	
E1	FLUSH	
E2-E5	NOT USED	

CONTROL BOARD LAYOUT



DEVICES

A1 - 20 MHz Osc.
A2 - SN74LS74
A3 - SN74LS00
A4 - SN74LS00
A5 - SN74LS367
A6 - SN74LS04
A7 - SN74LS367

B1 - SN74LS590
B2 - SN74LS590
B3 - SN74LS08

C1- SN74LS688
C2 - SN74LS688
C3 - SN74LS00

D1 - SN74LS374
D2 - SN74LS374

E1 - SN74LS590
E2 - SN74LS590
E3 - SN74LS590
E4 - SN74LS20

F1 - SN74LS688
F2 - SN74LS688
F3 - SN74LS688
F4 - SN74LS04

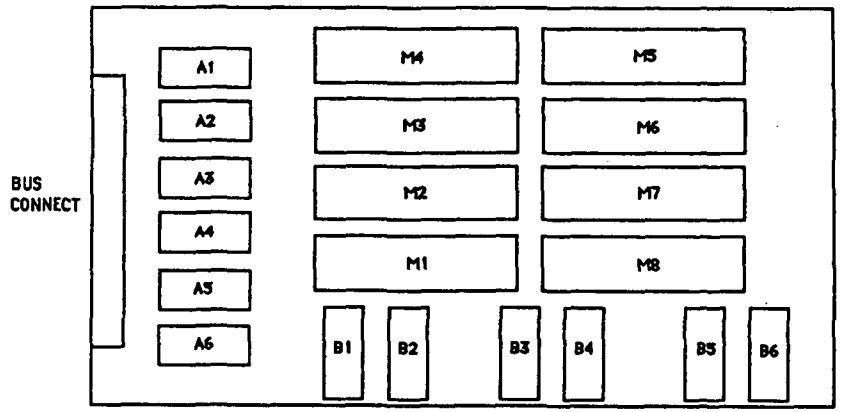
G1 - SN74LS590
G2 - SN74LS590
G2 - SN74LS590
G4 - SN74LS00

H4 - SN74LS00

I2 - SN74LS32
I3 - 7342
I4 - SN74LS04

$\overline{\text{PULSE}}$ INDICATES THE START OF INCOMING DATA

SHIFTER BOARD LAYOUT



DEVICES

A1 - SN74LS590 (low byte)
A2 - SN74LS590
A3 - SN74LS590 (high byte)
A4 - SN74LS590
A5 - SN74LS367
A6 - SN74LS00

B1 - SN74LS08
B2 - SN74LS00
B3 - SN74LS32
B4 - SN74LS32
B5 - SN74LS32
B6 - SN74LS32

M1 - M8 - IDT72104

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E: SUPPLEMENTAL INFORMATION REQUIRED FOR DIGITAL HARDWARE PROGRAMMING

THIS PAGE INTENTIONALLY LEFT BLANK

1. I/O Port Map

Address	Function Code	Port
0	16	Channel 0 delay
1	16	Channel 1 delay
2	16	Channel 2 delay
3	16	Channel 3 delay
4	16	Channel 4 delay
5	16	Channel 5 delay
6	16	Channel 6 delay
7	16	Channel 7 delay
8	16	Bits per bin count
9	16	Bins per scan
10	16	Shift register length
11	16	Reset and Mode control
12	16	Threshold value
13	16	Channel enable
0	0	Read FIFO data

2. Program/Control Codes

Channel 0 to 7 delay is programmed with 65536-(desired delay)

Bits per bin is programmed with 65536-255-desired bits

Bins per scan is programmed with 65535-desired bin count

Shift register length is programmed with 65536-(desired length divided by 2)

Reset and mode control port:

Reset is controlled by W1 (bit 0)
- 0 no reset
- 1 reset

Mode is controlled by W9 (bit 8)
- 0 single scan
- 256 continuous scan

Combined together
- 0 no reset and single scan
- 257 reset and continuous scan

To operate in single scan mode, hold the mode in the single scan state, pulse momentarily to the continuous state then back to single scan state. One scan of data will be acquired then the control logic waits for continuous mode to be set again. When operating in the continuous mode, the

hardware will begin looking for another image as soon as the FIFO is loaded with the bin values from the previous scan. If the FIFO is not read completely by the time the next scan is found the FIFO contents will be lost.

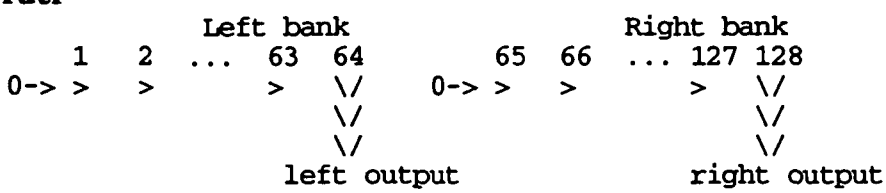
Threshold is programmed directly. The value programmed is the number of pulses in the right bank of the shift register which will trigger the logic to detect the image center.

The channel enable turns on or off individual channels. A one programmed in the corresponding channel position will enable the pulses into to the programmable delay. A zero will inhibit data from that channel. This feature is used to inhibit data from a channel connected to a bad PMT, amplifier, or discriminator. A failure of the circuitry after the edge detector output cannot be inhibited. (A failure in this circuitry is not likely.)

Channel	Bit position	Enabling Value
0	0	1
1	1	2
2	2	4
3	3	8
4	4	16
5	5	32
6	6	64
7	7	128

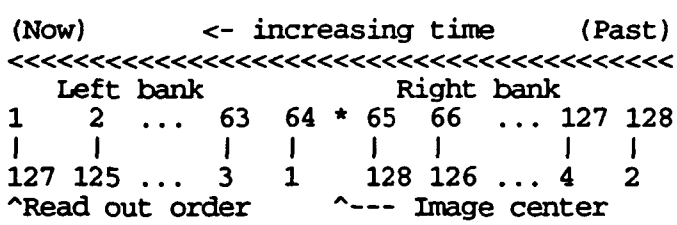
255 enables all channels, 0 disables all channels on board.

3. Bin Readout Order



Data is read out from the left bank output (middle tap) and right bank output (end tap) as the data is shifted right through the shift register. 0 is shifted into the left input and right input as the data is read out.

4. Bin Number in Order of Increasing Memory Address



Sequence of bin numbers as stored in FIFO: 64 128 63 127 ... 2 66 1 65

To order the data into an array with time increasing from the 1st element to the last element (time increasing from left to right), a Pascal software loop would look like this:

```
{ source and destination are arrays[1..128] of integer }
      x:=1;
      for i:=1 to 64 do
      begin
            destination[i] := source[x+1];
            destination[i+64] := source[x];
            x:=x+2
      end;
```

To order the data into an array with the bin number corresponding to the array element number:

```
      x:=1;
      for i:=64 downto 1 do
      begin
            destination[i] := source[x];
            destination[i+64] := source[x+1];
            x:=x+2;
      end;
```

5. Example of Output Data Rate Determination

Given:

Number of bins per scan = 128
Shift register length (Max) = 65536 samples
Max raw pulse rate = 1,000,000 per second average
Sample rate = 640 kHz
Prescaler divides by 10
Max scan rate = 64 Hz

Then:

Max samples per bin = 65536/128 = 512
Prescaled pulse rate = 1,000,000/10 = 100,000 = 1e5
Max number of pulses per bin = 512*1e5*(1/640 kHz) = 80
Max number of pulses from 8 channels = 80*8 = 640
Highest bin count expected = 640 (from above)

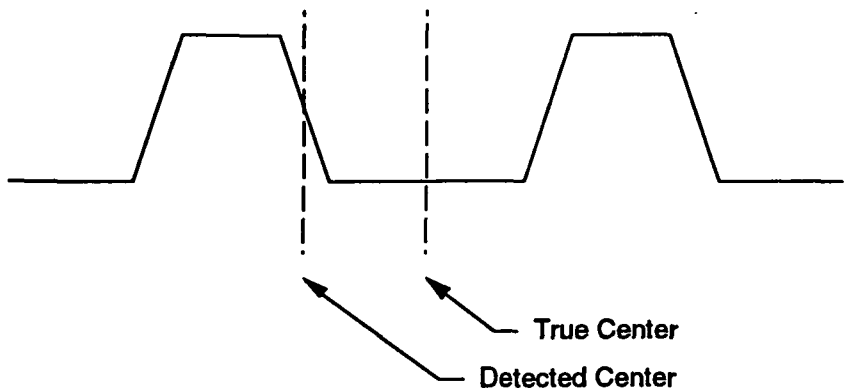
For constant high intensity image (never happens):
Max pulses in scan = 65536*1e5*(1/640 kHz)*8 = 81920
Max count in each half of shift register = 81920/2 = 40960

Data rate into computer = 64 Hz * 128 bins = 8192 words per Sec

Data rate from 8 boards = 8192 * 8 = 65536 words per Sec

6. Limitations Imposed by Profiles with Dead Time Between Groups of Pulses

If dead time exists between groups of pulses, some profiles will not center properly. Because the centering is based on the equality of pulses contained within the left and right shift register banks, a profile that has a period of inactivity will not trigger center until the active portion reaches the center tap. These profiles have the same pulse count in each bank for certain periods of time as the profile propagates through the shift register.



7. Limitations Associated with Low Intensity Image Profiles

If the total pulse count of an image is less than twice the programmed threshold certain images will not center properly. Suppose the threshold is programmed for 100 pulse counts. If the profile has 150 counts, when the first 100 reach the right bank, the left side then has 50 counts, the threshold will be crossed (because right >= threshold) and centering is enabled. Immediately center will be detected because the center criteria has be satisfied (right >= left). This will occur for images which start out at a high pulse density and have a decreasing number of pulses with time.

```
Decoding logic for CAMAC interface
 12/10/90 ver1 initial
 03/24/91 ver2 spare output driven high

 U6 on lpr6.sch

| pal22v10
|| INPUTS
  1 : clk       ,| clock notused
  2 : f2        ,| function code 2
  3 : f4        ,| function code 4
  4: s2true     ,| buffered strobe 2
  5 : f16       ,| function code 16
  6 : a1        ,| address 1
  7 : a2        ,| address 2
  8 : a4        ,| address 4
  9 : a8        ,| address 8
  10: ntrue     ,| board selected
  11: s1true    ,| buffered strobe 1
  13: f1        ,| function code 1

|| OUTPUTS
  23: port0to7en,| enable for 0-7 decoder
  22: port8     ,| bit/bin          (active low)
  21: port9     ,| bin/scan         (low)
  20: port10    ,| shift reg depth  (low)
  19: port11    ,| reset and mode   (low)
  18: port12    ,| ldthres          (high)
  17: porte13   ,| channel enable port (low)
  16: spare     ,| spare driven high
  15: readoe    ,| read buffer output enable
  14: fiford    ,| fifo rd pulse

| active-low: port0to7en, port[8..11], porte13, readoe, fiford

|| f[8] is dont care
 readfifo = (f[16,4,2,1] == 0) & (a[8,4,2,1] == 0)
 setfiford = (readfifo) & ntrue & s1true | turn on at S1
 resetfiford = s2true | turn off at S2
 fiford = setfiford # (fiford & resetfiford')
 readoe = readfifo & ntrue | enable buffers during n
 spare = 1

|| writeen is a pulse the width of S1
 writeen = (f[16,4,2,1] == 8 ) & ntrue & s1true | f(16) is code for write

 port0to7en = ( a[8,4,2,1] == 0xxxb  ) & writeen
 port8      = ( a[8,4,2,1] == 8      ) & writeen
 port9      = ( a[8,4,2,1] == 9      ) & writeen
 port10     = ( a[8,4,2,1] == 10     ) & writeen
 port11     = ( a[8,4,2,1] == 11     ) & writeen
 port12     = ( a[8,4,2,1] == 12     ) & writeen
 porte13    = ( a[8,4,2,1] == 13     ) & writeen

|vectors: (
 initialize
check fiforead
 display (f[16,4,2,1])d," ",(a[8,4,2,1])d," ",ntrue,s1true,s2true,"  ",readoe,"
... ",fiford
|*v 'f[16,4,2,1]' 3,'a[8,4,2,1]' 3,ntrue,s1true,s2true 3,readoe,fiford;
 set s2true=1
 set fiford=0 | reset fiford
|*d
```



```
 test f[16,4,2,1]=0; a[8,4,2,1]=0;ntrue=0;s1true=0;s2true=0;clk=0
 test f[16,4,2,1]=0; a[8,4,2,1]=0;ntrue=1;s1true=0;clk=0
 test f[16,4,2,1]=0; a[8,4,2,1]=0;ntrue=1;s1true=1;clk=0
 test f[16,4,2,1]=0; a[8,4,2,1]=0;ntrue=1;s1true=0;clk=0
 test f[16,4,2,1]=0; a[8,4,2,1]=0;ntrue=1;s2true=1;clk=0
 test f[16,4,2,1]=0; a[8,4,2,1]=0;ntrue=1;s2true=0;clk=0
 test f[16,4,2,1]=0; a[8,4,2,1]=0;ntrue=0;clk=0
|*o
check writes
 display (f[16,4,2,1])d," ",(a[8,4,2,1])d," ",ntrue,s1true,"  ",port0to7en,port8
... ,port9,port10,port11,port12,porte13
|*n 'f[16,4,2,1]' 3,'a[8,4,2,1]' 3,ntrue,s1true 3,port0to7en,port8;
|*v port9,port10,port11,port12,porte13;
|*d
 test f[16,4,2,1]=8; a[8,4,2,1]=0 ;ntrue=1;s1true=1;clk=0
 test f[16,4,2,1]=8; a[8,4,2,1]=7 ;ntrue=1;s1true=1;clk=0
 test f[16,4,2,1]=8; a[8,4,2,1]=8 ;ntrue=1;s1true=1;clk=0
 test f[16,4,2,1]=8; a[8,4,2,1]=9 ;ntrue=1;s1true=1;clk=0
 test f[16,4,2,1]=8; a[8,4,2,1]=10;ntrue=1;s1true=1;clk=0
 test f[16,4,2,1]=8; a[8,4,2,1]=11;ntrue=1;s1true=1;clk=0
 test f[16,4,2,1]=8; a[8,4,2,1]=12;ntrue=1;s1true=1;clk=0
|end)
```

```
clock generator
u111 on lpr5.sch
 12/10/90 ver 1 initial
 12/20/90 ver 2 changed wr and rd to active high
 12/30/90 ver 3 changed to 9x clock

 pal22v10
 INPUTS
  1 :clk      ,     | tied to 23 externally
  2 :sclken   ,     | enables sclk
  3 :reset    ,     | resets counter
  13:oscin    ,     | 9x sample clock from oscillator
 OUTPUTS
  23:sclk9x   ,     | buffered 9x sample clock
  22:contsclk ,     | continuous sample clock
  21:wr       ,     | write clock
  20:rd       ,     | read clock
  19:sclk     ,     | sample clock
  18:         ,     |
  14..17:q[3..0]    | 4 bit counter outputs

 Registers:   clk // q[3..0],wr,rd,sclk,contsclk

 active-low: reset

 Map:  Q[3..0] -> Q[3..0]
 ( n -> n+1, (n/=15 & n/=0)  & reset'
   n -> 8,   n==0  & reset'
   n -> 0,   n>=15 # reset
 )
 ensclk = (wr & sclken') # (q2 & sclken)
 wr = ensclk
 rd = ensclk'
 sclk = ensclk'
 contsclk = q2'
 sclk9x = oscin    | just a buffer

Vectors:
( initialize
  Display clk,(Q[3..0])d," ",sclken," ", (wr,rd,sclk,contsclk,q0,q1,q2,q3)c
*v clk, [3], 'Q[3..0]',sclken 2, wr, rd, sclk, contsclk, q0,q1,q2,q3;
  set q[3..0]=0
*d
  test clk
  Test reset=1; clk
  Test reset=0; sclken=1; clk=20(0,1)
  test reset=0; sclken=0; clk=20(0,1)
  Test reset=0; sclken=1; clk=20(0,1)
*o
  Test reset=0; sclken=1; clk=10(0,1)
*n clk;
  display (clk)c
*d
  test clk=3(0,1)
 End )
```

```
Center finder and counter control
  u90 on lpr3.sch
  12/10/90 ver 1 initial
  12/20/90 ver 2 changed usage of readout in enable signals
  12/30/90 ver 3 changed to 9x clock input.  Extra clock time allows
           all 8 channels to be scanned before next rising edge of
           contsclk. Only change is the use of spulse.
           spulse now inhibits counters so no counting occurs during
           extra clock cycle

| pal22v10
|  INPUTS
|  1 : clk      ,| sclk9x
|  2 : rmux     ,| right mux output
|  3 : lmux     ,| left mux output
|  4 : cmux     ,| channel data mux output
|  5 : spulse   ,| sample clock pulse
|  6 : contsclk ,| continuous sclk ( not used )
|  7 : accen    ,| accumulator enable
|  8 : readout  ,| readout memory occuring
|  9 :
|  10:
|  11:
|  13:
|  OUTPUTS
|  23: enr      ,| enable right counter
|  22: dirr     ,| dir right counter
|  21: enl      ,| enable left counter
|  20: dirl     ,| dir left counter
|  19: q0       ,| mux address counter qa
|  18: q1       ,| qb
|  17: q2       ,| qc
|  16:
|  15:
|  14:

| Registers:   clk // q[2..0]

| active-low: accen, spulse, readout, enr, enl

| Map:  Q[2..0] -> Q[2..0]
| ( n -> n+1, (n/=7) &   spulse' & accen
|   n -> 0,   ((n==7) #  spulse) & accen
|   n -> n,                        accen'
| )
| enables only look at other input if not reading out
| spulse occurs after all channels have been counted, inhibit during spulse
| enr  = spulse' & (rmux ## (lmux & readout')) | enable if rmux<>lmux
| enl  = spulse' & (lmux ## (cmux & readout')) | enable if cmux<>lmux
| dirr = lmux # readout | force up if readout
| dirl = cmux # readout


|vectors:
|( initialize
| display (clk, spulse)c, (q[2..0])d
|*v clk 3, spulse 3, 'q[2..0]';
|*d
| test spulse=1; accen=1; clk=1(0,1)
| test spulse=0; clk=8(0,1)
| test spulse=1; clk=1(0,1)
| test spulse=0; clk=8(0,1)
| test spulse=0; clk=3(0,1) | check that spulse resets counter anytime
```



```
| test spulse=1; clk=1(0,1)
| test spulse=0; clk=8(0,1)
| test spulse=0; accen=0; clk=3(0,1) | test enable holds count
|*o
| display spulse,readout,rmux,lmux,cmux,"  ",enr,dirr,enl,dirl
|*n spulse,readout,rmux,lmux,cmux 3,enr,dirr,enl,dirl;
|*d
| test spulse,readout,rmux,lmux,cmux
|end)
```

```
Counter control for bin and bit/bin counters
  u110 on lpr5.sch
  12/22/90 ver2 Addition of cload signal

| pal22v10
| INPUTS
  1 : clk,        | contsclk
  2 : binrco,     | bin counter rco output
  3 : bbinrco,    | bits/bin rco output
  4 : readout,    | readout occuring
|  5 :
|  6 :
|  7 :
|  8 :
|  9 :
|  10:
|  11:
|  13:

| OUTPUTS
  23: bincnttc,   | bin counter terminal count
|  22:
  21: bbintc,     | bits per bin terminal count
  20: bbinld,     | load bits per bin counter
  19: cload       | load bin counter
|  18:
|  17:
|  16:
|  15:
|  14:

| active-low: "all"
| registers: clk // bincnttc, bbintc
| bincnttc = binrco        | register binrco to make it synchronous
| bbintc   = bbinrco       | register bbinrco to make it synchronous
| bbinld   = bbintc # readout'  | load when not readout or tc is reached
| cload    = readout'           | hold loaded when not readout
| vectors: (
| end)
```

```
STATE MACHINE CONTROLLER
  u109 on lpr5.sch
  12/10/90 ver1 initial
  12/28/90 ver2 added clr0 state, changed clra to be with readout on
  12/28/90 ver3 changed clk to sclk removed spulse waits
  12/28/90 ver4 changed clrb to not wait until contmode
  12/30/90 ver5 added clr0 back in to let output regs preload

PAL22V10
 INPUTS
  1:clk,          | sclk
  2:rgtth,        | right >= threshold
  3:rgtl,         | right >= left
  4:spulse,       | sample clock pulse ( not used )
  5:bincnttc,     | bin count terminal count
  6:bbintc,       | bits per bin terminal count
  7:fbincnttc,    | fifo output bin count terminal count ( not used )
  8:contmode,     | continuous mode
  9:sreset,       | resets state machine

 OUTPUTS
 23:q0, |accen,    | accumulator enable
 22:q1, |accclr,   | accumulator clear
 21:q2, |readout,  | shift register readout occurring
 20:q3, |lrsel,    | left/right* select on output mux
 19:q4, |fifowr,   | fifo wr pulse
 18:
 17:q5, |reset,    | reset to fifo
 16:q6, |chxclr,   | channel x clear, clears all channels and shift reg RAM
 15:q7  |sclken    | enables sample clock
 14:

active-low: bincnttc, fbincnttc, bbintc, spulse, sreset
active-low: q0, q1, q2, q4, q5, q6
        accen, accclr, readout, fifowr, reset, chxclr
registers:
  clk // q[0..7]

procedure: sreset, q[0..7]
    (accen, accclr, readout, lrsel, fifowr, reset, chxclr, sclken)

( states:
   begin    =   01000011b,  | wait for contmode to say go
   shifting=    10000001b,  | waiting for threshold cross
   acquire =    10010001b,  | waiting for center, lrsel=1 creates unique state
   clr0     =   01000101b,  | let clock run one cycle to preload output regs
   clra     =   01100110b,  | clear with readout on, reset fifo
   binread =    10100011b,  | read a bin
   selleft =    00110010b,  | select left accumulator output
   wrleft  =    00111010b,  | write it to fifo
   selright=    00100010b,  | select right accumulator output
   wrright =    00101010b,  | write it to fifo
   clrb     =   01100010b   | clear accum after each bin read

   restart.
            -> begin       | just go to begin after reseting

   begin.
     contmode?  -> shifting | go to shifting if contmode
                -> begin    | wait for contmode to go active
   shifting.
     rgtth?  -> acquire     | look for center after threshold reached
```



```
             -> shifting    | wait here for right count to cross threshold
   acquire.
     rgtl?   -> clr0        | center found, co read bins
             -> acquire     | wait for center
   clr0.
             -> clra        | keep clock enabled one extra sycle to
                            | allow left and right registers to load
                            | with data to be counted and flush out
                            | data causing center detect

   clra.                    | disable sclk, then read bins
             -> binread
   binread.
     bbintc? -> selleft     | when bbintc is reached bin has been read
             -> binread
   selleft.                 | read left counter
             -> wrleft
   wrleft.                  | write it to fifo
             -> selright
   selright.                | read right counter
             -> wrright
   wrright.                 | write it to fifo
              -> clrb
   clrb.                    | clear counters
     bincnttc?  -> begin    | go get another scan when all bins read
     bincnttc'? -> binread  | go get another bin
)

vectors:
{
  display (rgtth, rgtl, spulse, bincnttc, bbintc, contmode)c," ",(clk, q[0..7])
  :c
*v [3], rgtth, rgtl, spulse, bincnttc, bbintc, contmode 5, clk;
*v accen, accclr, readout, lrsel, fifowr, reset;
*v chxclr, sclken;
*d
  test sreset=1; clk
  set sreset=0
  test bincnttc=0; contmode=0; clk=2(0,1)
  test contmode=1; clk=2(0,1)
  test rgtl=1; clk | test that r>l is ignored until r>th
  test rgtl=0; clk | it went away now
  test clk
  test rgtth=1; clk
  test clk=5(0,1)
  test rgtl=1; clk | test that r>l is seen because r>th
  test rgtl=0; rgtth=0; clk=5(0,1)
  test spulse=1; clk          | start binread
  test spulse=0; clk=3(0,1)
  test bbintc=1; clk=8(0,1)   | bin done
  test spulse=1; bbintc=0; clk          | go do another bin
  test spulse=0; clk=3(0,1)
  test bbintc=1; clk=8(0,1)   | bin done
  test spulse=1; bbintc=0; clk          | go do another bin
  test spulse=0; clk=3(0,1)
  test bbintc=1; clk
  test bbintc=0; clk=8(0,1)   | bin done
  test bincnttc=1,0; clk        | go back to start
  test clk=5(0,1)
end)
```

```
CAMAC bus interface logic
 u5 on lpr6.sch
 12/10/90 ver1 initial version
 12/21/90 ver2 active low statement was accidently commented out, fixed
 12/28/90 ver3 made bincnttc active low like it should be

 pal22v10
 INPUTS
  1 : clk          ,  clock not used
  2 : s1true       ,  strobe pulse 1
  3 : s2true       ,  strobe pulse 2
  4 : z            ,  z
  5 : c            ,  c
  6 : bincnttc     ,  bin count terminal count into fifo shows data ready
  7 : w8           ,  write bus bit 8
  8 : port11       ,  port 13 enable to latch w8
  9 : ntrue        ,  board select
  10: readoe       ,  readoe from other pld
  11: fbincnttc    ,  bincnttc out of fifo shows done reading
  13: hreset       ,  hardware initiated reset
  23: w0           ,  write bus bit 0

 OUTPUTS
  22: lreset       ,  latched w0 bit for sreset
  21:
  20: xgate        ,  x output to o.c. gate
  19: qgate        ,  q output to o.c. gate
  18: lgate        ,  look at me (LAM) output to o.c. gate
  17: contmode     ,  continuous/oneshot* mode
  16: sreset       ,  software reset output
  15: dataready    ,  fifo data ready (rs f/f output)
  14:

active-low:   z, c, readoe, port11, sreset, hreset, bincnttc, fbincnttc

        q =    set     # (   q      & reset' )
 dataready = (bincnttc) # (dataready & readoe')
 xgate = 1                | high all the time
 qgate = fbincnttc  | indicates when last value read
 lgate = ntrue' & dataready | turn off lgate during n
 contmode is an output bit on port11
 contmode = (port11 & w8) #          | transparant latch enable
            (port11' & contmode ) #  | hold output
            (w8 & contmode)          | prevent glitch in transition
 w0 = 0 ?? 0  | disable output because w0 is an input
 lreset = (port11 & w0) #          | transparant latch enable
          (port11' & lreset ) #    | hold output
          (w0 & lreset)            | prevent glitch in transition

 sreset = (c & s2true) # hreset # lreset

vectors: (
 display port11,w8," ",contmode
 test port11,w8=2(0..3)
end)
```

APPENDIX F: STREHL INTENSITY RATIO SELECTION CODE

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*************************************************************************
Selector.c

This file will determine the files that we will use to process our data.

*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <OSUtils.h>

void input(void);
void calculate(void);
void output(void);

    FILE    *gRawData1, *gData;
    char    gFilename1[255];
    float   gStrehl;
    double  gArray1[138],gArray2[138];

main()

{
    int i,n;

    printf("Enter the number of files you wish to evaluate\n");
    scanf("%d",&n);

    for(i=1;i<=n;i++)
    {
    input();
    calculate();
    output();

    fclose(gRawData1);
    fclose(gData);
    }

}

void input()
{

float   factor;

    /*  Prompt user for the filename of the old data    */

    printf("Enter the name of the file you wish to evaluate\n");
    scanf("%s",gFilename1);

    /* Open a file to read data */
```

```
    gRawData1=fopen(gFilename1,"r");

    /* Open a file to read data */
    gData=fopen("The List","a");


}


void calculate()
{
    int         i,j,k,inc;
    int         n=256;
    char        words1[255],words2[255], holder1[20],holder2[20];
    char        numholder1[20],numholder2[20];
    float       total1,midtotal1;
    double      bin1[140],bin2[140];

        i=j=k=0;
        total1=midtotal1=0;

        for(i=0;i<=136;i++)
        {
            if(i<=8)
            {
                fgets(words1,128,gRawData1);
            }
            else if (i>=100)
            {
                fgets(holder1,4,gRawData1);
                fgets(numholder1,8,gRawData1);

                bin1[i-8]=atof(holder1);
                gArray1[i-8]=atof(numholder1);
                total1 = total1 + gArray1[i-8];
            }
            else
            {
                fgets(holder1,3,gRawData1);
                fgets(numholder1,8,gRawData1);

                bin1[i-8]=atof(holder1);
                gArray1[i-8]=atof(numholder1);
                total1 = total1 + gArray1[i-8];
            }

        }

        k=28;

        midtotal1 = midtotal1 + gArray1[64];

        for(j=1;j<=k/2;j++)
```

```
        {
                midtotal1 = midtotal1 + gArray1[64-j];
                midtotal1 = midtotal1 + gArray1[64+j];
        }
        gStrehl=midtotal1/total1;
}


void output()
{
    long        seconds;
    DateTimeRec Date;
    int         j,n;

    GetDateTime(&seconds);
    Secs2Date(seconds,&Date);

    fprintf(gData,"Run Conducted on %2d/%2d/%2d at %2d:%2d\n",Date.month,Date.day,Date

    if(gStrehl>=0.8)fprintf(gData,"Good Data from %10s\t with Strehl Intensity Ratio o

}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G: FAST FOURIER TRANSFORM PROCESSING CODE

THIS PAGE INTENTIONALLY LEFT BLANK

```
#include <math.h>

#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void four1(data,nn,isign)
float data[];
int nn,isign;
{
    int n,mmax,m,j,istep,i;
    double wtemp,wr,wpr,wpi,wi,theta;
    float tempr,tempi;

    n=nn << 1;
    j=1;
    for (i=1;i<n;i+=2) {
        if (j > i) {
            SWAP(data[j],data[i]);
            SWAP(data[j+1],data[i+1]);
        }
        m=n >> 1;
        while (m >= 2 && j > m) {
            j -= m;
            m >>= 1;
        }
        j += m;
    }
    mmax=2;
    while (n > mmax) {
        istep=2*mmax;
        theta=6.28318530717959/(isign*mmax);
        wtemp=sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi=sin(theta);
        wr=1.0;
        wi=0.0;
        for (m=1;m<mmax;m+=2) {
            for (i=m;i<=n;i+=istep) {
                j=i+mmax;
                tempr=wr*data[j]-wi*data[j+1];
                tempi=wr*data[j+1]+wi*data[j];
                data[j]=data[i]-tempr;
                data[j+1]=data[i+1]-tempi;
                data[i] += tempr;
                data[i+1] += tempi;
            }
            wr=(wtemp=wr)*wpr-wi*wpi+wr;
            wi=wi*wpr+wtemp*wpi+wi;
        }
        mmax=istep;
    }
}

#undef SWAP
```

```
#include <math.h>

void realft(data,n,isign)
float data[];
int n,isign;
{
    int i,i1,i2,i3,i4,n2p3;
    float c1=0.5,c2,h1r,h1i,h2r,h2i;
    double wr,wi,wpr,wpi,wtemp,theta;
    void four1();

    theta=3.141592653589793/(double) n;
    if (isign == 1) {
        c2 = -0.5;
        four1(data,n,1);
    } else {
        c2=0.5;
        theta = -theta;
    }
    wtemp=sin(0.5*theta);
    wpr = -2.0*wtemp*wtemp;
    wpi=sin(theta);
    wr=1.0+wpr;
    wi=wpi;
    n2p3=2*n+3;
    for (i=2;i<=n/2;i++) {
        i4=1+(i3=n2p3-(i2=1+(i1=i+i-1)));
        h1r=c1*(data[i1]+data[i3]);
        h1i=c1*(data[i2]-data[i4]);
        h2r = -c2*(data[i2]+data[i4]);
        h2i=c2*(data[i1]-data[i3]);
        data[i1]=h1r+wr*h2r-wi*h2i;
        data[i2]=h1i+wr*h2i+wi*h2r;
        data[i3]=h1r-wr*h2r+wi*h2i;
        data[i4] = -h1i+wr*h2i+wi*h2r;
        wr=(wtemp=wr)*wpr-wi*wpi+wr;
        wi=wi*wpr+wtemp*wpi+wi;
    }
    if (isign == 1) {
        data[1] = (h1r=data[1])+data[2];
        data[2] = h1r-data[2];
    } else {
        data[1]=c1*((h1r=data[1])+data[2]);
        data[2]=c1*(h1r-data[2]);
        four1(data,n,-1);
    }
}
```

```
static float sqrarg;
#define SQR(a) (sqrarg=(a),sqrarg*sqrarg)
#include <math.h>

void Amp(data1,data2,nn)
float data1[];
float  data2[];
int nn;
{
    int     k;

    for(k = 2; k < nn+2; k++)           /* Get Amplitude of Difference FFT  */
        {
            data2[k-2] = sqrt(SQR(data1[k-2])+SQR(data1[k-1]));
        }


}
```

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <OSUtils.h>

void FileRead(data1,data2,ioname)
FILE    *ioname;
float   data1[];
int data2[];

{
    /* Read in data into zero based array */

    int     k, holder1, numholder1;
    char    words[180];

    for(k = 0; k < 138;k++)
    {
        if(k < 9)                   /* Strip out Header */
        {
            fgets(words, 180, ioname);
        }
        else                        /* Read in Data */
        {
            fscanf(ioname,"%d%d",&holder1,&numholder1);
            data1[k-9] = numholder1;
            data2[k-9] = holder1;
        }
    }
}
```

```
#include <math.h>

void LoPass(data1,nn,cutoff)
float data1[];
int nn,cutoff;
{
    int k;

    for(k = 0; k < nn; k++)                    /* Low Pass Filter*/
    {
        if (k < cutoff )
        {
        data1[k]= data1[k];
        }
        else
        {
        data1[k]= 0.0;
        }
    }
}
```

```
static float sqrarg;
#define SQR(a) (sqrarg=(a),sqrarg*sqrarg)

void MyDeconvlv(data1,data2,nn,ans)
float data1[],data2[],ans[];
int nn;
{
    int     i;
    float   mag2;

    for (i = 2; i < nn+2; i+=2)
    {
        mag2 = SQR(data2[i-2]) + SQR(data2[i-1]);

        if(mag2 == 0.0)
        {
        ans[i-2]=0.0;       /* Real Part */
        ans[i-1]=0.0;       /* Imaginary Part */
        }
        else
        {
        ans[i-2]=((data1[i-2]*data2[i-2])-(data1[i-1]*data2[i-1]))/(mag2*nn);   /* Rea
        ans[i-1]=((data1[i-1]*data2[i-2])+(data1[i-2]*data2[i-1]))/(mag2*nn);       /*
        }
    }

}
```

```
#include <math.h>

void NormArea(data1,data2,nn)
float data1[],data2[];
int nn;
{
    int     k;
    float   total;

    total = 0.0000;


            for(k=0;k< nn;k++)              /* Accumulates the Input Data   */
            {
                    total = total + data1[k];
            }

            for(k=0;k< nn;k++)              /* Normalizes the Input Area to Unity   */
            {
                    data2[k] = data1[k]/total;
            }

}
```

```
#include <math.h>

void NormPeak(data1,data2,nn,isign)
float data1[],data2[];
int nn,isign;
{
    int     inc, iflag1,iflag2,k;
    double  total1,total2;

    total1 = total2 = 0.0000;

        if(isign != 1)
        {
            for(k=0;k < nn-1;k++)               /* Finds the Peak of the Input Data */
            {
                if(data1[k+1] > data1[k])
                {
                    total1 = total1 + data1[k];

                }
            }
            for(k=0;k < nn-1;k++)               /* Normalizes to the Peak   */
            {
                    data2[k] = data1[k]/total1;

            }
        }
        else
        {
            for(k=2;k < nn+2;k+=2)              /* Finds the Peak of the Input Data */
            {
                if(data1[k] > data1[k-2])
                {
                    total1 = total1 + data1[k];
                }

                if(data1[k+1] > data1[k-1])
                {
                    total2 = total2 + data1[k+1];
                }

            }

            for(k=2;k < nn; k+=2)               /* Normalizes to the Peak   */
            {
                    data2[k-2] = data1[k-2]/total1;
                    data2[k-1] = data1[k-1]/total2;

            }


        }
}
```

```
#include <math.h>

void Pad(data1,data2, nn)
float data1[],data2[];
int nn;
{
    int k;

    for(k = 0; k < 2*nn; k++)                    /* Pad with Zeros */
    {
        if(k<nn/4)
        {
        data2[k]= 0.0;
        }
        else if (k<(5*nn)/4)
        {
        data2[k]= data1[k-nn/4];
        }
        else
        {
        data2[k]= 0.0;
        }
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H: DIGITAL PROFILER SOFTWARE CODE

Code for the following Pascal software units is included in this appendix:

Unit Name	Description of Contents
DPGlobals.p	Global variables
DPDialogs.p	Procedures for handling of dialog boxes
sercom	Procedures for communicating through RS-232 serial port
instrintf	Procedures for communicating with signal processing electronics
dpgpib	Procedures for communicating with National Instruments GPIB board
DPHistogram.p	Procedures for plotting of histogram in main window
DPFile.p	Procedures for opening and saving files
DPPrint.p	Procedures for printing of histogram
DPVideo.p	Procedures for presenting live video from CCD in the main window
DPUtilities.p	Procedures for handling of menu commands
DPMain.p	Procedures for software interface

THIS PAGE INTENTIONALLY LEFT BLANK

```
unit DPGlobals;
{ 07/22/91 added const and type sections from instrinst}
{          added monum, mopos var}

interface
{the following const and type sections came from instrinst}
  const
    maxlen = 512; {maximum length of scan array}
    maxnumspeeds = 4; {4 speeds 0..3}
    numspeeds = 3; {0..3 = 4 }
    maxnumboards = 7; {number of digital boards}
    samplerate = 640000.0; {digital electronics sample rate}

  type
    scanarray = array[0..maxlen] of longint; {for summed data}

  const
    windoid = 128;  { resource ID for floating pallate window }
    mainWindow = 256;     { Resource ID for main window }
    AppleMenu = 128;      { Resource ID }
    FileMenu = 129;       { Resource ID }
    EditMenu = 130;       { Resource ID }
    DataMenu = 131;       { Resource ID }
    ImageMenu = 132;      { Resource ID }
    MOMenu = 133;         { Resource ID }
    UtilityMenu = 134;    { Resource ID }
    SpeedMenu = 135;      { Resource ID }

    CanceID = 128;  { Cancel Printing dialog }
    CommandID = 129;      { Send command dialog }
    EnableID = 130; { Enable/disable dialog }
    ObjectID = 131; { Object Info dialog }
    GridEnableID = 132;  { Grid Enable/disable dialog }
    NumberID = 1001;      { Get a single number dialog }
    QuitID = 1004;        { Save Changes dialog        }
    ParameterID = 1005;  { Parameter info... dialog   }
    ResolutionID = 1007; { Resolution Dialog    }
    CalibrateID = 133;{ Calibration Dialog    }
    AboutID = 1006;       { About Digital Profiler... dialog  }
    SorryID = 1008;       { Not programmed yet dialog   }
    MiscID = 1011;        { Misc info dialog            }
    ErrorID = 1020;       { Error dialog                }
    CCoffeeID = 128;      { Resource ID for cursor }
    CLogoID = 129;        { resource ID for cursor }

    Creator = 'DPRO';     { For documents           }
    FileType = 'DIMG';    { File Type (Digital IMaGe)   }
    maxWindows = 25;      { max. number of open windows }
    menuHeight = 40;      { height of menubar           }
    scrollWidth = 16;     { width of scrollbar          }

    inputCICNID = 129;    { resource ID for inputCICN }
    inputHiliteCICNID = 131;
    inICONID = 131;       { resource ID for zoomIn ICON }
    outICONID = 132;      { resource ID for zoomOur ICON }

    upArrow = $7E;        { key codes for arrow keys }
    downArrow = $7D;
    leftArrow = $7B;
    rightArrow = $7C;

    moResDefault = '100';          { default values for stage resolutions }
    irResDefault = '100';
```

```
    focusResDefault = '100';
    horizResDefault = '100';

  type
    wInfoHandle = ^wInfoPtr;     { a handle is a pointer to a pointer }
    wInfoPtr = ^WindowInfo;  { a pointer points to something in memory }
    WindowInfo = record        { information specific to each window }
      end;

  var
    coffee: CCrsrHandle;              { handle for cursor }
    logo: CCrsrHandle;                { handle for cursor }
    code: integer;
    errcode: byte;                    { used for error reporting }
    dataArray: scanArray;             { integer array of data }
    sortdata: scanarray;              { data after sorting}
    wRect: rect;                      { text window rectangle}
    done: Boolean;                    { flag to exit program }
    errno: Integer;                   { generic number used for errors }
    fileName: Str255;                 { current file name for data }
    dataSaved: Boolean;               { true if file has been saved }
    gotEvent: Boolean;                { for wait next event }
    theEvent: EventRecord;            { record returned by WaitNextEvent }
    dragRect: Rect;                   { limit for window dragging }
    hasFileName: Boolean;             { TRUE if data has filename }
    textisvisible: Boolean;           { true if text window is visible }
    iBeam: Cursor;                    { I-beam cursor }
    iCurs: CursHandle;                { I-beam cursor handle }
    inputCICN: CIconHandle;           { handle for CICN }
    inputHiliteCICN: CIconHandle;     { handle for CICN }
    inICON: Handle;                   { handle for zoom in icon }
    outICON: Handle;                  { handle for zoom in icon }
    inputRect: Rect;                  { location rectangle for inputCICN }
    inRect: Rect;                     { rectangle for zoom in icon }
    outRect: Rect;                    { rectangle for zoom out icon }
    cursorRgn: RgnHandle;             { region for WaitNextEvent }
    numWindows: Integer;              { number of windows }
    threshold: Longint;               { discriminator threshold }
    numberofScans: LongInt;           { number of scans to acquire }
    numberofBins: LongInt;            { number of data bins }
    numberofboards: longint;          { number of digital boards. used by dialog box}
    numboards: integer;               { used by hardware}
    myPt: Point;                      { random point }
    time: LongInt;                    { time since startup }
    savePort: GrafPtr;                { generic grafPtr }
    totalCounts: Longint;             { total number of counts in scan }
    theDialog: DialogPtr;             { generic dialogPtr }
    theItem: Integer;                 { generic Item selected }
    theWindow: WindowPtr;             { generic windowPtr }
    vRefNum: Integer;                 { volume number for current file }
    watch: Cursor;                    { watch cursor }
    wCurs: CursHandle;                { handle for watch cursor }
    whichControl: ControlHandle;      { handle for activated control }
    wInfo: wInfoHandle;               { handle for currently active window record }
    whichWindow, myWindow: WindowPtr;
    statusWindow: WindowPtr;  { pointer for pallete window }
    scanrate: array[0..maxnumspeeds] of integer;

{ stepper motor variables }
    monum: integer; { number of currently selected MO }
    xnum: integer;{ X stage position of currently selected MO }
    ynum: integer;{ Y stage position of currently selected MO }
    wnum: integer;{ W stage position of currently selected MO }
```

```
    {mopos maps menu selection item to wheel position}
    mopos: array[1..20] of integer;
    {xpos maps MO number to X position}
    xpos: array[1..20] of integer;
    {ypos maps MO number to Y position}
    ypos: array[1..20] of integer;
    {wpos maps MO number to wheel position}
    wpos: array[1..20] of longint;
    moRes: Str255; { MO Stage resolution Stage W}
    horizRes: Str255;     { horizontal stage resolution Stage X }
    focusRes: Str255;     { focus stage resolution Stage Y}
    irRes: Str255;        { Image rotator resolution  Stage Z}
    XDefault: Str255;     { horizontal stage position Stage X }
    YDefault: Str255;     { focus stage position   Stage Y}
    WDefault: Str255;          { MO Stage position  Stage W}
    scanSpeedValue: Str255;    { current scan speed }

{ image variables }
    observatory: Str255; { name of observatory }
    objectName: Str255;  { name of current object }
    RAvalue: Str255;     { right ascension value }
    decValue: Str255;    { declination value }
    magFilter: Str255;   { current magnification/filter }
    videoRect: Rect;          { location of CCD camera picture}

{ PICT file variables }
    PICTCount: LongInt;  { current size of picture }
    globalRef: Integer;  { fileSystem reference number }
    newPICTHand: PicHandle; { used for storing picture to be saved }

{ screen drawing variables }
    plotColor: RGBColor; { color used for plotting histogram }
    gridColor: RGBColor; { color for histogram grid }
    blackRGB: RGBColor;  { black }
    whiteRGB: RGBColor;  { white }
    plotFrame: Rect;     { rectangle used for plotting data }
    gRefNum: Integer;    { ref number for MTV driver }

{ real time acquisition variable }
    imageNumber: integer;          { image number to save to disk }
    realtimeflag: boolean;         { flag to tell event loop that we are getting date i
 RT}
    saverealtimeflag: boolean;     { tells whether or not to save data }
    realtimerefnum: integer;  { where to save on disk }
    realtimefilename: Str255; { name to save realtime data under }
implementation

end.
```

THIS PAGE INTENTIONALLY LEFT BLANK

```
unit DPDialogs;

interface
  uses
    SANE, DPGlobals;

  procedure DoOutline (theDialog: DialogPtr);
  procedure DoCenterDialog (theDialog: DialogPtr);
  procedure DoError (errno: Integer);
  function DoMisc (theError: Str255): Integer;

implementation

  procedure DoOutline; {(theDialog : DialogPtr)}
{ ••• Procedure DoOutline outlines the first item (default) in dialog list ••• }

    var
      box: Rect;
      itemType: Integer;
      item: Handle;

  begin
    GetDItem(theDialog, 1, itemType, item, box);
    PenSize(3, 3);
    InsetRect(box, -4, -4);
    FrameRoundRect(box, 16, 16);
  end;

  procedure DoCenterDialog; {(theDialog : DialogPtr);}
{ ••• Procedure DoCenterDialog centers given dialog on screen ••• }
    var
      temp: Extended;
      tempRect: Rect;

  begin
    tempRect := theDialog^.portRect;
    temp := ((screenBits.Bounds.Bottom - screenBits.Bounds.Top) - (tempRect.Bottom -
 tempRect.Top)) div 3;
    tempRect.Top := Num2Integer(temp);
    tempRect.Left := ((screenBits.Bounds.Right - screenBits.Bounds.Left) - (tempRect.Ri
 tempRect.Left)) div 2;
    MoveWindow(theDialog, tempRect.Left, tempRect.Top, TRUE);
    ShowWindow(theDialog);
    SelectWindow(theDialog);
    SetPort(theDialog);
  end; { Procedure DoCenterDialog }

  procedure DoError; {(errno : Integer)}
{ ••• Procedure DoError displays error dialog with error number ••• }

    var
      savePort: GrafPtr;
      theDialog: DialogPtr;
      theError: Str255;
      theItem: Integer;

  begin
    GetPort(savePort);
    SetCursor(arrow);
    NumToString(errno, theError);
    ParamText(theError, '', '', '');
    theDialog := GetNewDialog(ErrorID, nil, Pointer(-1));     { get dialog box }
    SetPort(theDialog);
```

```
    DoOutline(theDialog);
    DoCenterDialog(theDialog);
    ModalDialog(nil, theItem);              { put dialog box up; get result  }
    DisposDialog(theDialog);                { get rid of dialog box          }
    SetPort(SavePort);
  end; { of DoError }

  function DoMisc; {(theErr : Str255)}
{  displays a dialog with the given  text RS }

    var
      savePort: GrafPtr;
      theDialog: DialogPtr;
      theItem: Integer;

  begin
    GetPort(savePort);
    SetCursor(arrow);
    ParamText(theError, '', '', '');
    theDialog := GetNewDialog(MiscID, nil, Pointer(-1));    { get dialog box }
    SetPort(theDialog);
    DoOutline(theDialog);
    DoCenterDialog(theDialog);
    ModalDialog(nil, theItem);              { put dialog box up; get result  }
    DisposDialog(theDialog);                { get rid of dialog box          }
    SetPort(SavePort);
    DoMisc := theItem;
  end; { of DoMisc }

end.
```

```
unit sercomm;
{simple serial interface routines}
{7/15/19 created}
{7/16/91 fixed serialwrite. start at s[1] not s[0]}
{7/18/91 move source code to ultimate digital profiler folder}
{ 8/7/91 added delay before serialwrite occurrs. something hangs the computer}
{***** receiving serial data with out reading it causes hang. Buffer location?}
{ overrun?}
{ 8/8/91 used new() to allocate space for buffer defined as array}
{          ^ didnt help}
{         disabled xon/xoff, seems to eliminate the hanging problem}
{         removed delay in serialwrite}

interface
  uses
    serial;

  procedure serialopen;
  procedure serialwrite (s: str255);
  function serialread (count: integer): str255;


implementation

  const
    buffersize = 256;
    inrefnum = -6;
    outrefnum = -7;

  type
    str255 = string[255];
    bufferdata = array[0..buffersize] of integer; {this makes it twice as big}

  var
    bufptr: ^bufferdata;
    bufferptr: ptr; {pointer to input buffer}
    mysershk: sershk; {serial handshake settings}
    serconfig: integer;
    err: oserr;

  procedure error (s: str255);
  begin
    writeln(s, err);
    halt;
  end;

  procedure serialopen;
  {open serial port for read and write}
    var
      x: integer;
  begin
    serconfig := baud9600 + stop10 + noparity + data8;
    bufferptr := newptr(buffersize);
    new(bufptr); {allocate space}
    with mysershk do
      begin
        fxon := 0; {disable xon/xoff, 1=enable}
        fcts := 0; {disable hardware handshake}
        xon := chr(17); {control q}
        xoff := chr(19); {control s}
        errs := 0; {errors that cause abort}
        evts := 0; {status changes that cause events}
        finx := 1; {xon.xoff input flow control flag}
```

```
      end;
    err := opendriver('.Aout', x);
    if err <> noerr then
      error('error opening Aout:');
    if x <> outrefnum then
      error('outrefnum not correct');
    err := opendriver('.Ain', x);
    if err <> noerr then
      error('error opening Ain:');
    if x <> inrefnum then
      error('inrefnum not correct');
    err := serreset(outrefnum, serconfig);
    if err <> noerr then
      error('error setting output config');
    err := serhshake(outrefnum, mysershk);
    if err <> noerr then
      error('error setting output handshake');
    err := serreset(inrefnum, serconfig);
    if err <> noerr then
      error('error setting input config');
    err := serhshake(inrefnum, mysershk);
    if err <> noerr then
      error('error setting input handshake');
(*  err := sersetbuf(inrefnum, bufferptr, buffersize);*)
    err := sersetbuf(inrefnum, pointer(bufptr), buffersize);
    if err <> noerr then
      error('error setting buffer');
  end;

{   procedure serialwrite (s: str255);}
  procedure serialwrite;
  {write string to serial port}
    var
      count: longint;
  begin
    count := length(s);
    err := fswrite(outrefnum, count, (@s[1])); {output}
    if err <> noerr then
      error('error outputting string');
  end;

{   function serialread (count: integer): str255;}
  function serialread;
  {read data from the serial port}
    var
      c: longint;
      d: str255;
  begin
    c := count;
    err := fsread(inrefnum, c, @d[1]);
    if err <> noerr then
      error('error inputting data');
    if c <> count then
      error('error in input data length');
    d[0] := char(count); {set length of string}
    serialread := d;
  end;

end.
```

```
unit instrintf;
{unit to interface to the digital profiler instrument}
{ 7/15/91 started adding code to call serial interface }
{ 7/18/91 made swapandadd take a length parameter}
{         converted to a unit}
{ 7/18/91 move the source code to the Ultimate digital profiler folder}
{ 7/19/91 added errorh called when scsi error occurs and halts }
{ 7/29/91 fixed some things. added enableallchannels/disable to interface}
{         moved sortdata to dpglobals, fixed sumarray stack poping}
{ 7/31/91 worked on deskewing code}
{         made sumdata and sortdata lontint arrays}
{         changed sortandadd to add int in rawdata to longint in sumdata}
{ 8/2/91  added calcskew to calculate skew as 1/64th scan with out measuring}
{ 8/5/91  added scanmaxave function reduce noise in qmax calc}
{         added peakpulserate function to return pulse rate of max bin}
{         added totalnumchsen and ischenabled }
{ 8/7/91  worked on deskew code}
{ 8/8/91  remove bins div 2 in initialization. div by 2 was wrong}
{         changed operation mode to continuous from single to avoid}
{          having to trigger each time}
{ 8/21/91 changed scan speeds to 8, 16, 25, 50}
{ 8/22/91 added dialog to scsigetnselect for displaying errors }
{ 8/23/91 initallboards calls initdiscriminators}
{         threshold in globals used for disc thresh}
{ 8/24/91 changed dthres to add 40 instead of 5}
{ 8/25/91 fixed and added bad scan test. if center bin is 0 then ignore scan}
{         made numboards work in acquire}


{$R-}
 {turn range checking off}

interface

  uses
    dpglobals,{ has some types and const used here}
    DPDialogs, { for error reporting }
    sercomm, {simple serial interface routines}
    scsi;

  var
    arraylen: integer; {length of data array read from board}
    scanspeednum: integer; {number of scan speed}
    ccrate: longint;    {scsi address of the CAMAC crate}
    lastscans: longint; {number of scans in last acquire}

  procedure cleararray (var x: scanarray);
  procedure startmotor;
  procedure stopmotor;
  procedure setmotorspeed (scanspeed: integer);
  function scantotal (s: scanarray): longint;
  function scanmax (s: scanarray): longint;
  function scanmaxave (s: scanarray): longint;
  procedure enablechannel (ch: integer);
  procedure disablechannel (ch: integer);
  procedure enableallchannels;
  procedure disableallchannels;
  procedure initallboards (scanspeed, bins: integer);
  procedure initdiscriminators (threshold: integer);
  procedure acquire (nscans: longint);
  procedure deskew;
  procedure calcdeskew;
  procedure initeverything;
```

```
  procedure initdefaults;
  procedure endprogram;
  function whatissrlength: longint; {returns value of srlength}
  function whatisdthresh: integer; {returns value of digital count threshold}
  function peakpulserate: real;
  function totalnumchsen: integer;
  function ischenabled (ch: integer): boolean;

implementation

  const
   {digital electronics ports}
    binsport = 9;
    bitsperbinport = 8;
    srlengthport = 10;
    dthreshport = 12;
    chenableport = 13; {channel enable, bit number = channel number}
    delayport = 0;     {delay ports are 0 to 7 for channel 0 to 7}
    controlport = 11;

    dutycycle = 0.50; {active scan time/total scan time}
    {dutyclcle must be < hardware duty cycle to achieve the desired scan rate}
    {hardware duty cycle is .66. for x image time it takes .5*x to read out data}
    {total time for cycle is x+.5x=1.5 x. so: x/1.5x = 0.66.}

  type
    cmdarray = packed array[0..12] of byte; {used to send SCSICMD}
    skewvaluetype = array[0..numspeeds, 0..maxnumboards, 0..7] of integer;
    rawarray = array[0..maxlen] of integer; {for raw data}

  var
    i: integer;
    lasterr: OSerr;
    lamdata: longint;
    scsitib: array[1..5] of SCSIInstr; {transfer instruction block(s) (TIB)}
    scsiblktib: array[1..2] of SCSIInstr; {tib for block read}
    cmd: cmdarray; {used for read write commands}
    blkcmd: cmdarray; {used for block command}
    rwdata: integer; {used to send and receive single data values}
    sumdata: scanarray; {summed data stored here}
    buffer: longint;{pointer to source/dest data}
    status, tempdata: integer;{values returned by 3929 status check}

    junk: integer; {allocate space for garbage word before rawdata array}
    rawdata: rawarray; {dest of data read from boards}
    {bnaf* is used for block transfers, holds slotn,address,fcode}
    bnafhi, bnaflo: array[0..maxnumboards] of byte; {for quick loading of commands}

    slotnumber: array[0..maxnumboards] of integer; {holds slot number of board}

    starttime, endtime: longint; {used to calculate execution times}

    skewvalue: skewvaluetype;

    skewfile: file of skewvaluetype; {disk file to hold skewvalue array}

    chenable: array[0..maxnumboards] of byte;{bit map of which channels are enabled}

    srlength: longint; {shift register length, global because two proc use it}
    dthresh: integer; {digital count threshold}
    zerodataerrcount: integer; {number of low center bin errors}
    dataerrcount: integer;     {number of negative number errors}
```

```
procedure errorh;
{common point where scsi errors end up}
begin
  writeln('error occured, program halted');
  halt;
end;

function whatissrlength: longint;
begin
  whatissrlength := srlength;
end;

function whatisdthresh: integer;
begin
  whatisdthresh := dthresh;
end;

function realtostr (x: real): string;
{convert real number to fixed point string}
{6:4 > x.xxxx }
  var
    v: string[6];
    i: integer;
begin
  v := stringof(x : 6 : 4);
  for i := 1 to length(v) do
    if v[i] = ' ' then
      v[i] := '0';
  realtostr := v;
end;

procedure byteswap (var x: integer);
{inline code to swap bytes of an integer value}
inline
  $205F, { move.l (a7)+,a0 ; pop ao get address of value}
  $3010, { move.w (a0),d0  ; move value to d0}
  $E058, { ror.w #8,d0     ; rotate right lower word of d0 8 bits}
  $3080; { move.w d0,(a0)  ; move do to (a0), store swaped value in x}

procedure longswap (var x: longint);
{swap the bytes in a longint resuling from the scsi transfer}
  var
    temp, upperword, lowerword: integer;
    lowerptr, upperptr: ptr;
begin
  upperword := hiword(x);
  lowerword := loword(x);
  byteswap(upperword);
  byteswap(lowerword);
  x := bsl(lowerword, 16) + upperword;
end;

procedure swapandadd (var s: rawarray; d: scanarray; num: integer);
{swaps bytes in s and adds num elements to d array}
{s is array of integers, d is array of longint }
{integer in s[x] is added to longint in d[x] }
{num is number of elements}
inline
  $7200, { moveq.l   #0,d1 ; clear d1 to clear out high bits}
         {                 ; next pop only loads lower word of d1}
  $321F, { move.w (a7)+,d1 ; pop number of array elements (word)}
  $5341, { subq #1,d1      ; loop count needs to be n-1 for n loops }
  $225F, { move.l (a7)+,a1 ; pop a1 get address of d}
```

```
    $205F, { move.l (a7)+,a0 ; pop ao get address of s}
    $7000, { moveq.l   #0,d0 ; clear d0 so high word is zero for loop }
  { loop: }
    $3018, { move.w (a0)+,d0 ; move s[a0] value to d0, inc a0}
    $E058, { ror.w #8,d0     ; rotate right lower word of d0 8 bits}
    $D199, { add.l d0,(a1)+  ; add swaped value to sum }
    $51C9, { dbf d1,-4 words(jump to loop }
    -8;    { displacement in bytes}


procedure cleararray (var x: scanarray);
{clear out the array x}
  var
    i: integer;
begin
  for i := 0 to arraylen - 1 do
    x[i] := 0;
end;

procedure sortarray (var s, d: scanarray);
{sort the array of data to correct for interleaving done by read from CAMAC}
{s sorted -> d}
  var
    i, x: integer;
begin
  x := 0;
  for i := 0 to (arraylen div 2) - 1 do
    begin
      d[i] := s[x + 1];
      d[i + 64] := s[x];
      x := x + 2;
    end;
end;

procedure startmotor;
{send commands on serial port to start scan mirror motor at 1 RPS}
begin
  serialwrite('s'); {stop motor}
  serialwrite(chr(13)); {<cr>}
  {next line: motor steps, no posistion maintainance}
  serialwrite('11d3 mr25000 1fsc0 1fsb0');
  serialwrite(chr(13)); {<cr>}
  serialwrite('1fsd0 mc a10 ad10 v1 ');
  serialwrite(chr(13)); {<cr>}
  serialwrite('g');
  serialwrite(chr(13)); {<cr>}
end;

procedure stopmotor;
{stop the scan motor}
begin
  serialwrite('s');
  serialwrite(chr(13)); {<cr>}
end;

procedure setmotorspeed (scanspeed: integer);
{set motor speed rev per second. scanspeed is number 0 to 3}
  var
    speed: real;
begin
  serialwrite('s'); {stop motor first}
  serialwrite(chr(13)); {<cr>}
  speed := scanrate[scanspeed] / 50; {50 mirror facets per revolution}
```

```
    writeln('setting speed to: ', realtostr(speed));
    serialwrite('v'); {send motor speed change command}
    serialwrite(realtostr(speed));{send value of speed}
    serialwrite(chr(13)); {<cr>}
    serialwrite('g'); {start it up again}
    serialwrite(chr(13)); {<cr>}
  end;

  procedure scsigetnselect;
  {get scsi bus, select crate}
  begin
    lasterr := SCSIGet;
    if lasterr <> noerr then
      begin
        writeln('Error getting bus:', lasterr : 3);
        errorh;
      end
    else {got bus ok}
      begin
        lasterr := SCSISelect(ccrate);
        if lasterr <> noerr then
          begin
            theItem := DoMisc('Unable to select SCSI Crate Controller. Check to see if
is on.');
            writeln('Error selecting crate:', ccrate, ' err:', lasterr);
            errorh;
          end;
      end;
  end;

  procedure scsigetbus;
  {get the scsi bus}
  begin
    lasterr := SCSIGet;
    if lasterr <> noerr then
      begin
        writeln('Error getting bus:', lasterr : 3);
        errorh;
      end;
  end;

  procedure scsiselecttarget;
  {select the CAMAC crate as the target device}
  begin
    lasterr := SCSISelect(ccrate);
    if lasterr <> noerr then
      begin
        writeln('Error selecting crate:', ccrate, ' err:', lasterr);
        errorh;
      end;
  end;

  procedure makescsitib (buffer: longint; bytecount: integer);
  {make transfer interface block for a scsi command}
  begin
    with scsitib[1] do
      begin
        scopcode := scnoinc;
        scparam1 := buffer; {destination for data read}
        scparam2 := bytecount; {byte count}
      end;
    with scsitib[2] do
      begin
```

```
        scopcode := scstop;
        scparam1 := 0;
        scparam2 := 0;
      end;
  end;

  procedure buildnaf (slot, address, fcode: integer; var nafhi, naflo: byte);
  {build high and low NAF bytes for 3929 commands}
    var
      naf: integer;
  begin
    { naf := bsl(slot,9)+bsl(address,5) + fcode; }
    naf := slot * 512 + address * 32 + fcode; {put number in proper fields}
    nafhi := bsr(naf, 8); {get high byte}
    naflo := band(naf, $FF); {get low byte}
  end;

  procedure buildsinglecmd (slot, address, fcode: integer);
  {build single CAMAC command block for 3929 controller}
    var
      nafhi, naflo: byte;
  begin
    buildnaf(slot, address, fcode, nafhi, naflo);
    {mode 0,0,0,0,tm1,ws2,ws1,ad - ws=01 for 16bit}
    cmd[0] := $09; {single camac operation command opcode}
    cmd[1] := 0;   {logical unit and reserved MUST be zero}
    cmd[2] := $0b;  {mode control, 16 bit words, q ignore, disable x response}
    cmd[3] := nafhi; {slot,address function code}
    cmd[4] := naflo;
    cmd[5] := 0;    {control byte always zero}
    buffer := ord(@cmd[6]); {buffer points to cmd[6]}
    makescsitib(buffer, 2); {transfer two bytes}
  end; {buildsinglecmd}

  procedure camwrite (slot, address, fcode, data: integer);
  {write data to specified slot, address,fcode}
    var
      er: integer;
      lasterr: oserr;
  begin
    if fcode < 16 then {its a read command, error}
      begin
        writeln('Error: read fcode sent to camwrite function');
        errorh;
      end
    else
      begin
        buildsinglecmd(slot, address, fcode);
        cmd[6] := band(data, $ff); {low byte first}
        cmd[7] := bsr(data, 8);
        scsigetnselect; {get bus and select crate}
        lasterr := SCSICMD(@cmd[0], 6); {send 'single' command to crate}
        if lasterr <> noerr then
          begin
            writeln('Error sending SCSICMD write:', lasterr : 3);
            errorh;
          end
        else {cmd ok}
          begin
            lasterr := SCSIWrite(@scsitib[1]); {now send the data to the crate}
            if lasterr <> noerr then
              begin
                writeln('Error doing SCSIwrite cmd:', lasterr : 3);
```

```
            errorh;
          end
        else {write ok}
          begin
            lasterr := SCSIComplete(status, tempdata, 100);
            if lasterr <> noerr then
              begin
                writeln('Error doing complete command:', lasterr : 3);
                errorh;
              end;
          end; {write ok}
      end; {cmd ok}
  end; {ok}
end; {camwrite}


function camread (slot, address, fcode: integer): integer;
{read data from slot, address, fcode}
  var
    er: integer;
    lasterr: oserr;
    readdata: integer;
begin
  if fcode >= 16 then {its a write command, error}
    begin
      writeln('Error: write fcode sent to camread function');
      errorh;
    end
  else
    begin
      buildsinglecmd(slot, address, fcode);
      cmd[6] := $d2; {1234 decimal}
      cmd[7] := $04; {set 6 and 7 to see if any result returned}
      scsigetnselect; {get bus and select crate}
      lasterr := SCSICMD(@cmd[0], 6);
      if lasterr <> noerr then
        begin
          writeln('Error sending camread command:', lasterr : 3);
          errorh;
        end
      else {cmd ok}
        begin
          if SCSIRead(@scsitib[1]) <> noerr then
            begin
              writeln('Error doing SCSIRread cmd');
              errorh;
            end
          else {read ok}
            begin
              lasterr := SCSIComplete(status, tempdata, 100);
              if lasterr <> noerr then
                begin
                  writeln('Error doing complete command:', lasterr : 3);
                  errorh;
                end;
            end; {read ok}
        end; {cmd ok}
    end; {ok fcode}
  camread := cmd[7] * 256 + cmd[6]; {data is sent low byte first, do a swap}
end; {camread function}

procedure camcmd (slot, address, fcode: integer);
{address a board, but there is no data}
```

```
    var
      er: integer;
      lasterr: oserr;
  begin
    buildsinglecmd(slot, address, fcode);
    makescsitib(0, 0); {overwrite scsitib with 0 and 0 data bytes}
    scsigetnselect; {get bus and select crate}
    lasterr := SCSICMD(@cmd[0], 6);
    if lasterr <> noerr then
      begin
        writeln('Error sending camread command:', lasterr : 3);
        errorh;
      end
    else {cmd ok}
      begin
        lasterr := SCSIComplete(status, tempdata, 100);
        if lasterr <> noerr then
          begin
            writeln('Error doing complete command:', lasterr : 3);
            errorh;
          end;
      end; {cmd ok}
  end; {camcmd function}

  procedure requestsense;
  {reads status values from 3929.  Refer to user manual for what the codes mean}
    var
      data: packed array[0..45] of byte;
      cmd: array[1..2] of longint;
      i: integer;
  begin
    for i := 0 to 45 do
      data[i] := 255;
    cmd[1] := $03000000; {request sense allocate 32 bytes (20hex)}
    cmd[2] := $20000000;
    makescsitib(ord(@data[0]), $20);
    scsigetnselect;
    lamdata := 1234567;
    if scsicmd(@cmd[1], 6) <> noerr then
      begin
        writeln('rs cmd err');
        errorh;
      end;
    if scsiread(@scsitib[1]) <> noerr then
      begin
        writeln('rs read err');
        errorh;
      end;
    if scsicomplete(status, tempdata, 10) <> noerr then
      begin
        writeln('rs complete err');
        errorh;
      end;
    writeln('rs stat,tempdta: ', status, tempdata);
    writeln('sense key codes: ', data[0] : 3, data[2] : 3, data[12] : 3, data[13] : 3);
  end; {requestsense}

  procedure testunitready;
  {verify that 3229 is capable of communicating}
  {uses variable called lamdata because code was copied from readlam}
    var
      lamdata: longint;
      lamcmd: array[1..2] of longint;
```

```
    i: integer;
  begin
    lamcmd[1] := $00000000; {test unit ready}
    lamcmd[2] := 0;
    makescsitib(ord(@lamdata), 0);
    lamdata := 1234567;
    scsigetnselect;
    if scsicmd(@lamcmd[1], 6) <> noerr then
      begin
        writeln('t cmd err');
        errorh;
      end;
    if scsiread(@scsitib[1]) <> noerr then
      begin
        writeln('t read err');
        errorh;
      end;
    if scsicomplete(status, tempdata, 10) <> noerr then
      begin
        writeln('t complete err');
        errorh;
      end;
    if (status <> 0) or (tempdata <> 0) then
      writeln('Testunitready error');
  end;

  function readlam: longint;
  {read bit mapped look at me register from 3929. bit0 = slot 1}
    var
      lamdata: longint;
      lamcmd: array[1..2] of longint;
      nafhi, naflo: byte;
  begin
    buildnaf(30, 12, 1, nafhi, naflo);
    {mode 0,0,0,0,tm1,ws2,ws1,ad - ws=00 for 24bit}
    cmd[0] := $09; {single camac operation command opcode}
    cmd[1] := 0;   {logical unit and reserved MUST be zero}
    cmd[2] := $09;  {mode control, 24 bit words, q ignore, disable x response}
    cmd[3] := nafhi; {slot,address function code}
    cmd[4] := naflo;
    cmd[5] := 0;    {control byte always zero}
    makescsitib(ord(@lamdata), 4); {4 bytes returned}
    scsigetnselect;
    lamdata := 1234567;
    if scsicmd(@cmd[0], 6) <> noerr then
      begin
        writeln('lam cmd err');
        errorh;
      end;
    if scsiread(@scsitib[1]) <> noerr then
      begin
        writeln('lam read err');
        errorh;
      end;
    if scsicomplete(status, tempdata, 4) <> noerr then
      begin
        writeln('lam complete err');
        errorh;
      end;
    longswap(lamdata); {reverse byte order}
{    lamdata := $00FFFFFF; this is to get activity when instrument is disconneted}
    readlam := lamdata;
  end;
```

```
procedure ksc3929init;
{initialize the kinetic system 3929 scsi controller}
begin
  {sequence copied from protosoft go event}
  {30 is controller slot}
  camwrite(30, 0, 17, 4); {set inihibit}
  camwrite(30, 0, 17, 1); {generate initialize cycle (z)}
  camwrite(30, 0, 17, 2); {generate clear cycle (c)}
  camwrite(30, 0, 17, 0); {clear inhibit}
 {writeln('control reg: ', camread(30, 0, 1));}
end; {3929init}

procedure initboardnaf;
{initialize arrays of slot, address, fcode to be used by block command}
{also initializes the slotnumber array}
  var
    address, fcode, i: integer;
begin
  slotnumber[0] := 2; {board 0 is slot 2}
  slotnumber[1] := 3;
  slotnumber[2] := 7;
  slotnumber[3] := 8;
  slotnumber[4] := 13;
  slotnumber[5] := 14;
  slotnumber[6] := 18;
  slotnumber[7] := 19;
  address := 0; {read address to read data array}
  fcode := 0;   {fcode to read data array}
  for i := 0 to numboards do
    begin
      buildnaf(slotnumber[i], address, fcode, bnafhi[i], bnaflo[i]);
    end;
end;

procedure initscsiblktib (dest: longint; count: integer);
{initialize the scsiblktib array (block command transfer interface block) }
{dest is address of destination}
{count is number of integers to read. bytes=2*count}
begin
  with scsiblktib[1] do
    begin
      scopcode := scnoinc;
      scparam1 := dest; {destination for data read}
      scparam2 := count * 2; {byte count}
    end;
  with scsiblktib[2] do
    begin
      scopcode := scstop;
      scparam1 := 0;
      scparam2 := 0;
    end;
  {initialize blkcmd command array.  }
  blkcmd[0] := $22;{block transf opcode. q-ignore, word size 16 bits}
  blkcmd[1] := 0;
  blkcmd[2] := $0b; {mode}
  blkcmd[3] := bnafhi[2]; {replaced when executing blockread}
  blkcmd[4] := bnaflo[2]; { '' }
  blkcmd[5] := 0; {hi byte of transfer count}
  blkcmd[6] := bsr(count * 2, 8);
  blkcmd[7] := band(count * 2, $ff); {low byte of transfer count}
  blkcmd[8] := 0;
  blkcmd[9] := 0; {these bytes are zero}
```

```
end;

procedure blockread (board: integer);
{block read from slot,address,fcode assigned to board in initboardnaf. }
{Store at destination initialized by initscsiblktib above}
{count is number of integer values to read}
begin
  blkcmd[3] := bnafhi[board];
  blkcmd[4] := bnaflo[board];
  scsigetnselect; {get bus and select crate}
  if SCSICMD(@blkcmd, 10) <> noerr then
    begin
      writeln('Error sending camread command in blockread:');
      halt;
    end;
  if SCSIRead(@scsiblktib[1]) <> noerr then
    begin
      writeln('Error doing SCSIRread cmd in blockread: ');
      halt;
      {requestsense;}
    end;
  if SCSIComplete(status, tempdata, 1) <> noerr then
    begin
      writeln('Error doing complete command:');
      halt;
    end;
end; {blockread}

function scantotal (s: scanarray): longint;
{calculate the total counts in the scan}
  var
    i: integer;
    total: longint;
begin
  total := 0;
  for i := 0 to arraylen - 1 do
    total := total + s[i];
  scantotal := total;
end;

function scanmaxave (s: scanarray): longint;
{return the average of the peak and adjacent bins}
  var
    i: integer;
    max: longint;
    bin: integer;
begin
  max := 0;
  for i := 0 to arraylen - 1 do
    if s[i] > max then
      begin
        max := s[i]; {save highest value}
        bin := i;    {save bin number}
      end;
  max := 0;
  for i := bin - 1 to bin + 1 do  {add up adjacent ones}
    begin
      if (i < 0) or (i > arraylen - 1) then
        max := max + s[bin] {add in middle if +/-1 are out of bounds}
      else
        max := max + s[i];
    end;
  scanmaxave := round(max / 3); {average them}
```

```
end;

function scanmax (s: scanarray): longint;
{return the highest count value contained in any bin}
  var
    i: integer;
    max: longint;
begin
  max := 0;
  for i := 0 to arraylen - 1 do
    if s[i] > max then
      max := s[i];
  scanmax := max;
end;

function peakpulserate: real;
{return photon pulse rate calculated from peak bin}
{pulse rate = 10 * (sample rate) * (pulses in bin)/(samples in bin)}
{*10 is because of the hardware prescaler}
  var
    peakonescan: real;
begin
  {peak in one scan is average of all enabled channels}
  peakonescan := (scanmax(sortdata) / lastscans) / totalnumchsen;
  peakpulserate := 10 * samplerate * peakonescan / (srlength div arraylen);
end;

procedure updateenable;
{send enable words to all boards}
  var
    board: integer;
begin
  for board := 0 to numboards do
    camwrite(slotnumber[board], chenableport, 16, chenable[board]);
    {enable channels}
end;

procedure enablechannel (ch: integer);
{channel 1 to 64 is on board 0 to 7, channel 0 to 7}
  var
    b, c: integer;
begin
  b := ((ch - 1) div 8); {board number}
  c := ((ch - 1) mod 8); {channel number}
  {set the corresponding bit in the enable bye}
  chenable[b] := bitor(chenable[b], bsl(1, c));
  updateenable;
end;

procedure disablechannel (ch: integer);
{channel 1 to 64 is on board 0 to 7, channel 0 to 7}
  var
    b, c: integer;
begin
  b := ((ch - 1) div 8);
  c := ((ch - 1) mod 8);
  {clear the corresponding bit in the enable bye}
  chenable[b] := bitand(chenable[b], bnot(bsl(1, c)));
  updateenable;
end;

procedure enableallchannels;
  var
```

```
    i: integer;
begin
  for i := 0 to 7 do
    chenable[i] := $ff;
  updateenable;
end;

procedure disableallchannels;
  var
    i: integer;
begin
  for i := 0 to 7 do
    chenable[i] := 0;
  updateenable;
end;

function numchsen (board: integer): integer;
{returns number of channels enabled for this board}
  var
    i, t: integer;
begin
  t := 0; {number of ch enabled}
  for i := 0 to 7 do {interrogate each channel}
    begin
      if btst(chenable[board], i) then {channel is enabled}
        t := t + 1;
    end;
  numchsen := t;
end;

function totalnumchsen: integer;
{returns total number of channels enabled on all boards}
  var
    board, x: integer;
begin
  x := 0; {clear count before starting}
  for board := 0 to numboards do {add em up}
    x := x + numchsen(board);
  totalnumchsen := x;
end;

function ischenabled (ch: integer): boolean;
{return true if channel is enabled}
  var
    b, c: integer;
begin
  b := ((ch - 1) div 8);
  c := ((ch - 1) mod 8);
  if btst(chenable[b], c) then
    ischenabled := true
  else
    ischenabled := false;
end;

procedure triggerboard (board: integer);
{trigger one board}
{******* a way to speed this up is to do a block write of the 2 words}
{then the overhead of camwrite is not suffered twice}
{or run in the continuous mode all the time and dont trigger boards}
  var
    i: integer;
begin
  {toggle from cont to single mode triggers one scan to be acquired}
```

```
{    i := camread(slotnumber[board], 0, 0); }
{read one value to clear lam}
    camwrite(slotnumber[board], controlport, 16, 256); {continuous mode}
    camwrite(slotnumber[board], controlport, 16, 0);   {single scan mode}
  end;

  procedure getboarddata (board: integer);
  {read data from the board restart it to acquire the next scan}
    const
      displaymode = false;
      mincount = 2;
  begin
    blockread(board); {read data from the board}
    {ignore scan if first or last read value is negative, or if center bin is 0}
    {bytes have not been un-swapped yet so test bit 7 for neg. instead of 15}
    {rawdata[0] is bin 64 (center bin).  if it is zero then scan was bad}
    {dont have to worry about swap for 0 because 0 swapped is still 0}
    if (btst(rawdata[0], 7)) or (btst(rawdata[arraylen - 1], 7)) then
      begin {data bad}
        if displaymode then
          begin
            byteswap(rawdata[0]); {so they display right}
            byteswap(rawdata[arraylen - 1]);
            writeln('Data error', rawdata[0], rawdata[arraylen - 1]);
          end;
        dataerrcount := dataerrcount + 1;
      end
    else
      begin {check two center bins for acceptable level}
        if (rawdata[0] < mincount) or (rawdata[arraylen - 1] < mincount) then
          begin
            zerodataerrcount := zerodataerrcount + 1;
          end
        else {no data errors}
          swapandadd(rawdata, sumdata, arraylen);
{    triggerboard(board); we are running continuous now}
      end;
  end;

  procedure triggerallboards;
  {trigger all boards to acquire data}
    var
      i, x: integer;
  begin
    for i := 0 to numboards do
      begin
        {reset board and clear lam before triggering}
        x := camread(slotnumber[i], 0, 0); {read one value to clear lam}
(*triggerboard(i); dont trigger if in continuous mode*)
      end;
  end;

  procedure clearalllams;
  {clear all lams on all boards}
    var
      i, x: integer;
  begin
    for i := 0 to numboards do
      begin
        {reset board and clear lam before triggering}
        x := camread(slotnumber[i], 0, 0); {read one value to clear lam}
      end;
  end;
```

```
  procedure setresetmode;
  {hold reset on all boards}
    var
      i: integer;
  begin
    for i := 0 to numboards do
      begin
        camwrite(slotnumber[i], controlport, 16, 1);{reset}
      end;
  end;

  procedure setcontinuous1ton;
  {set 1-7 boards to continuous mode}
    var
      i, x: integer;
  begin
    for i := 1 to numboards do
      begin
        camwrite(slotnumber[i], controlport, 16, 256); {rel reset and cont mode}
      end;
  end;

  procedure setcontinuous;
  {set all boards to continuous mode}
    var
      i, x: integer;
  begin
    for i := 0 to numboards do
      begin
        camwrite(slotnumber[i], controlport, 16, 256); {rel res, continuous mode}
      end;
  end;

  procedure initdiscriminators (threshold: integer);
  {initialize the discriminator board thresholds}
  {threshold in millivolts}
    var
      discboard: integer;
      dslot: array[0..3] of integer;
  begin
    dslot[0] := 4; {first board in slot 4}
    dslot[1] := 9;
    dslot[2] := 15;
    dslot[3] := 20;
    for discboard := 0 to 3 do
      begin
        camcmd(dslot[discboard], 0, 26); {set internal mode bit}
        camwrite(dslot[discboard], 0, 17, threshold - 10); {boards wants -10}
      end;
  end;

  procedure initallboards (scanspeed, bins: integer);
  {initialize all boards to read data}
  {scanspeed is the number of the scanrate}
    var
      bitsperbin: integer;
{dthresh:integer; is global to this unit}
{srlength: longint;  its a global to this unit now}
{may be greater than 2^15}
      hbitsperbin, hsrlength, hbins, hdthresh: integer; {hardware values derived}
      board: integer;
      scantime: real;
```

```
    slot, i: integer;
    sampletime: real;
    x: integer;
  const
    srlengthmax = 65534; {cant have sr longer than this}
    binsmax = 512;
    thoffset = 5; {add this to dark pulses to get threshold}

begin
  sampletime := 1.0 / samplerate; {time represented by one sample}
  scantime := dutycycle / scanrate[scanspeed]; {active time of one scan}
  srlength := trunc(scantime / sampletime); {calculate approximate length}
  bitsperbin := srlength div bins;
  {now recalc srlength so its an integer multiple of bitsperbin}
  srlength := longint(bitsperbin) * bins; {force a longint calculation}
  if srlength > srlengthmax then
    begin
      srlength := srlengthmax; {cant have length greater than hardware}
      writeln('Shift register length > max. internal error');
    end;
  {threshold for one channel}
  dthresh := trunc((srlength / (samplerate / 100)) + thoffset); {100 dark PPS}
  {now calc values needed by hardware}
  hbitsperbin := 65281 - bitsperbin;
  hsrlength := 65536 - (srlength div 2);
  hbins := 65535 - bins;
  for board := 0 to numboards do
    begin
      slot := slotnumber[board];
      camwrite(slot, chenableport, 16, 0); {disable all channels}
      camwrite(slot, bitsperbinport, 16, hbitsperbin);
      camwrite(slot, srlengthport, 16, hsrlength);
      camwrite(slot, binsport, 16, hbins);
      hdthresh := dthresh * numchsen(board);
      if hdthresh = 0 then
        hdthresh := 10000; {make it big if all channels are disabled}
      camwrite(slot, dthreshport, 16, hdthresh);
      for i := 0 to 7 do
        begin {write delay for channel}
          camwrite(slot, i, 16, integer(65535 - skewvalue[scanspeed, board, i]));
        end;
      camwrite(slot, controlport, 16, 1);   {reset}
      camwrite(slot, controlport, 16, 0);   {relase reset, single scan mode}
      camwrite(slot, chenableport, 16, chenable[board]); {enable channels}
      x := camread(slot, 0, 0); {read data to clear lam}
    end; {for board}
  initdiscriminators(threshold); {initialize threshold}
end; {initall}

procedure acquirex (nscans: longint);
{acquire nscans number of scans from CAMAC crate}
{one scan is a read from 8 boards, as counted by detecting consecutive}
{reads from one reference board.  Provision is made for the reference board}
{dieing.  We wont wait for ever, but the actual scan count will be wrong}
{enbles board 0 and waits for its lam, then enables the other boards}
  var
    scans: longint; {count of number of scans}
    i, timecount: integer;
    lamdata: longint;
    firsttime: boolean; {see if first read}
    refboard: integer;  {board used to count scans}
  const
    timeout = 2000;
```

```
  begin
    lastscans := nscans; {save for peakrate calculation}
    dataerrcount := 0;
    zerodataerrcount := 0;
    setresetmode;{hold boards in reset so they dont trigger}
    clearalllams; {clear all LAMs on boards}
{    setcontinuous; set all boards to continuous mode, enable then to acquire}
    camwrite(slotnumber[0], controlport, 16, 256);{rel reset,cont mode board 0 only}
   {now loop until we have read nscans of data}
    firsttime := true;
    scans := 0;
    nscans := nscans * 8;
    cleararray(sumdata);
    repeat
      timecount := 0;
      repeat
        lamdata := readlam; {this is probably time consuming}
        timecount := timecount + 1;
      until (lamdata <> 0) or (timecount > timeout);
      if timecount > timeout then
        begin
          writeln('Timeout error waiting for LAM');
          scans := nscans + 1; {cause repeat loop to end}
        end
      else {no timeout}
        begin
          if firsttime then {see which board is active}
            begin
              refboard := -1;
              i := 0;
              repeat
                if btst(lamdata, slotnumber[i] - 1) then
                  begin
                    refboard := i;
                  end;
                i := i + 1;
              until (i > numboards) or (refboard <> -1);
              if i > numboards then
                writeln('didnt find a board ', lamdata);
              if refboard = -1 then
                refboard := 7;
              setcontinuous1ton;{set boards to cont mode, enable then to acquire }
              firsttime := false; {only do first time once}
            end; {first time}
          for i := 0 to numboards do {check all boards for active level}
            begin
            {lam bits: 0 = slot 1}
              if btst(lamdata, slotnumber[i] - 1) then {board has data}
                begin
                  getboarddata(i); {get the data from the board and add to sum}
                  if i = refboard then {inc by 8 and zero lower bits}
                    scans := bitand(scans + 8, $FFF8)
                  else {not refboard, inc by one so we eventually get out of here}
                    scans := scans + 1;
                end;
            end; {for numboards}
        end; {no timeout}
    until (scans >= nscans);
    sortarray(sumdata, sortdata);
    writeln('Data errs (neg,0):', dataerrcount : 5, zerodataerrcount : 5);
  end; {acquire}

  procedure acquire (nscans: longint);
```

```
    {acquire nscans number of scans from CAMAC crate}
      var
        scans: longint; {count of number of scans}
        i, timecount: integer;
        lamdata: longint;
      const
        timeout = 2000;
    begin
      lastscans := nscans; {save for peakrate calculation}
      dataerrcount := 0;
      zerodataerrcount := 0;
      setcontinuous;
      clearalllams; {clear all LAMs on boards}
(*    triggerallboards; *)
{trigger them to acquire data}
      cleararray(sumdata);
      scans := 0;
      {now loop until we have read nscans of data}
      repeat
        timecount := 0;
        repeat
          lamdata := readlam;
          timecount := timecount + 1;
        until (lamdata <> 0) or (timecount > timeout);
        if timecount > timeout then
          begin
            writeln('Timeout error waiting for LAM');
            scans := nscans + 1; {cause repeat loop to end}
          end
        else {no timeout}
          begin
            for i := 0 to numboards do
              begin
              {lam bits: 0 = slot 1}
                  if btst(lamdata, slotnumber[i] - 1) then {board has data}
                    begin
                      getboarddata(i); {get the data from the board and add to sum}
                      scans := scans + 1;
                    end;
              end; {for numboards}
          end; {no timeout}
      until (scans >= nscans);
      sortarray(sumdata, sortdata);
      writeln('Data errors : ', dataerrcount, zerodataerrcount);
    end; {acquire}

    procedure displaydata (s: scanarray);
      var
        i, j, k: integer;
        scale: integer;
      const
        scalefactor = 40;
    begin
      scale := trunc(scanmax(s) / (scalefactor));
      if scale = 0 then
        scale := 1;
      writeln('scale : ', scale);
      for i := 0 to arraylen - 1 do
        begin
          k := (s[i] div scale);
          if k > scalefactor + 5 then
            k := scalefactor + 5;
          write(i : 3, ' ');
```

```
        for j := 1 to k do
          write(' ');
        writeln('*');
      end;
  end;

  procedure deskewchannel (scanspeed, board, cha, chb: integer);
  {figure best relative delay between cha and chb on one board}
  {cha and chb are 0 to 7}
  {reads delay value for cha from table to start.  writes value to table}
  {for delay on chb that is minimum relative skew. }
  {minimum skew is measured by maximumizing the ratio (peak/sum) of the }
  {scan that has just cha and chb enabled}
  {cha skew value must already be in the skewvalue[] array}
    var
      skew, bestskew: integer;
      chaskew, chbskew: integer;
      q, qmax: real; {ratio of peak to sum}
      coursestep: integer;
      limit: integer;

    procedure peak;
    {embedded procedure for code used by course and fine loop}
    {get data and check for peak q}
    begin
      camwrite(slotnumber[board], chb, 16, 65535 - skew); {write the delay}
      acquire(20); {get a bunch so its sort of smooth, one rev on scan mirror}
      q := scanmaxave(sumdata) / scantotal(sumdata); {indicates how effective the deske
is}
      if q > qmax then
        begin
          bestskew := skew;
          qmax := q;{save highest value}
        end;
      writeln('skew,q,bestskew ', skew, q : 8 : 5, bestskew);
    end;

  begin
    showtext;
    chaskew := skewvalue[scanspeed, board, cha]; {get skew for cha}
    disableallchannels;
    enablechannel(board * 8 + cha + 1); {convert to ch 1 to 64}
    enablechannel(board * 8 + chb + 1);
    initallboards(scanspeed, arraylen); {calulates srlength for us}
(*    coursestep := 10;*)
    coursestep := srlength div 128; {do in steps of 1 bin}
    camwrite(slotnumber[board], cha, 16, 65535 - chaskew); {write ch a skew}
    qmax := 0;
    {loop from (expected-1/64th) to (expected+1/64th) delay}
    skew := (6 - chb) * (trunc(srlength / dutycycle) div 64);
    limit := (8 - chb) * (trunc(srlength / dutycycle) div 64);
    while (skew <= limit) do
      begin {course loop}
        peak; {get data and check for peak}
        skew := skew + coursestep;
      end; {course while loop}
    qmax := 0;
    writeln('doing fine');
    for skew := bestskew - coursestep to bestskew + coursestep do {fine steps}
      begin {fine loop}
(*    peak;*)
      end; {fine loop}
    skewvalue[scanspeed, board, chb] := bestskew;
```

```
  end;

  procedure writeskewvalues;
  {write the skewvalue array to disk}
  begin
    open(skewfile, 'skewvalues'); {create/open the file}
    write(skewfile, skewvalue); {write the array to the file}
    close(skewfile); {close it we are done}
  end;

  procedure readskewvalues;
  {read the skewvalue array from disk}
  begin
    iocheck(false);
    open(skewfile, 'skewvalues'); {create/open the file}
    read(skewfile, skewvalue); {read the array from the file}
    if ioresult <> 0 then
      writeln('scanfile not there');
    close(skewfile); {close it we are done}
    iocheck(true);
  end;


  procedure deskew;
  {perform the deskewing on all channels by measuring}
  {results saved in skewvalue array}
    var
      scanspeed: integer;
      boardnum: integer;
      channelnum: integer;
  begin
    writeln('performing measurement of de-skew values');
    {clear array of skewvalue}
    for scanspeed := 0 to numspeeds do
      for boardnum := 0 to numboards do
        for channelnum := 0 to 7 do
          skewvalue[scanspeed, boardnum, channelnum] := 0;
{     for scanspeed := 0 to numspeeds do}
    for scanspeed := 3 to 3 do {just do one for now}
      begin
        setmotorspeed(scanspeed); {set speed to a speed}
{     for boardnum := 0 to numboards do}
        for boardnum := 2 to 2 do {just do one for now}
          begin
          {start with channel with least delay (7) }
            for channelnum := 6 downto 0 do
              begin
                writeln('Deskewing ', scanspeed, boardnum, channelnum);
                deskewchannel(scanspeed, boardnum, 7, channelnum);
              end; {for channel}
          end; {for boardnum}
      end; {scanspeed}
    writeskewvalues; {save em to disk}
  end; {deskew}


  procedure calcdeskew;
  {calculates array of skew values instead of measuring them}
  {results saved in skewvalue array}
    var
      scanspeed: integer;
      boardnum: integer;
      channelnum: integer;
```

```
    t: integer;
  begin
    writeln('Performing calculation of de-skew values (not measurement)');
    for scanspeed := 0 to numspeeds do
      begin
        initallboards(scanspeed, arraylen); {calculates srlength for us}
        for boardnum := 0 to numboards do
          begin
            for channelnum := 0 to 7 do
              begin
                {delay is 1/64th scan between each channel}
                t := round(((srlength / dutycycle) / 64)); {delta skew}
                skewvalue[scanspeed, boardnum, channelnum] := (7 - channelnum) * t;
                writeln('calc ', scanspeed, boardnum, channelnum, (7 - channelnum) * t)
              end; {for channel}
          end; {for boardnum}
      end; {scanspeed}
    writeskewvalues; {save em to disk}
  end; {deskew}

  procedure initeverything;
  {probably should only call this once (because of serialopen)}
  begin
    readskewvalues; {read the skewvalue array from disk}
    initboardnaf; {initialize array of board naf s}
    initscsiblktib(ord(@rawdata) - 2, arraylen + 1);{-2,+1 makes room for garbage word}
    cleararray(sumdata);
    ksc3929init; {initialize the controller}
    requestsense;
    testunitready;
    writeln('tested unit ready');
{    enableallchannels;}
    initallboards(scanspeednum, arraylen); {x scans/sec initialize the boards}
    writeln('initialized the boards');
    serialopen; {open the serial port for read/write}
    writeln('opened the serial driver');
    startmotor;
    writeln('started motor');
    setmotorspeed(scanspeednum); {set speed to a speed}
    writeln('set motor speed');
    lastscans := 1; {set in acquire}
  end;

  procedure endprogram;
  {call when ending the program}
  begin
    stopmotor;  {dont leave the motor spinning}
  end;

  procedure initdefaults;
  {initialize then defaults, these are variables that can be changed}
  begin
    scanrate[0] := 6; {scanrate 0 is 6 hz}
    scanrate[1] := 12;
    scanrate[2] := 25;
    scanrate[3] := 50;
    ccrate := 5;  {SCSI sddress of crate selected by thumbweel on 3929}
    arraylen := 128; {number of elements in array, 0..arraylen-1}
    numboards := 7; {8 boards addressed 0 to 7}
    scanspeednum := 3; {select highest scans/sec}
    threshold := 125; {125 mV}
  end;
```

```
end.
```

```
unit DPGPIB;
{ unit with GPIB (ieee-488) bus interface routines}
{ 7/19/91 created }
{ 8/24/91 added array for x,y,w to hold position of axes for each MO}
{ 8/25/91 added correction factor for each axis incase of it gets bumped}
{ 8/30/91 added FudgeFactor for each axis from Calibration routine}
{ 9/3/91 added x,y,w coordinates for each MO}
{ 9/16/91 added close video statement, should the gpib board not be present}

interface
  uses
    DPDialogs, { for error roporting }
    DPVideo,   { for video stuff }
    DPGlobals; { for good stuff }

  procedure opengpib;
  procedure writegpib (s: string);
  procedure selectmo (num: integer);
  procedure DoGoHome;

implementation


  const
    ibfind = 25; {function code for ibFIND}
    ibsic = 12;  {ibSIC}
    ibrd = 6;    {ibRD}
    ibwrt = 14;  {ibWRT}
    gpibslotname = 'gpib4'; {board is in slot 4. this name is passed to the }
                            {driver to tell it where the board is installed}
    klinger = 'MC4';{device address of klinger controller}

  type
    statusblock = record {status parameter block for GPIB driver}
        ibsta: integer;
        iberr: integer;
        ibret: integer;
        ibcnt: longint;  {this needs to be a 10 byte block}
      end;

    sblkptr = ^statusblock; {poitner type to status block above}

    gpibblock = record {GPIB parameter block for the driver}
        statusptr: sblkptr; {pointer to status block}
        id: integer;
        controlvar: integer;
        iobuf: ptr;
        iocount: longint;   {the record needs to be 16 bytes}
      end;

    gpibblockptr = ^gpibblock; {pointer to GPIB block}

  var
    gpib: gpibblock; {allocate space for a gpib block}
    status: statusblock;  {allocate space for a status block}
    gpibpointer: gpibblockptr; {pointer to a gpib block}
    refnum: integer;
    err: integer;
{bdname: packed array[1..7] of char;}
    bdname: string[10];
    boardid: integer;
{    dvname: packed array[1..7] of char;}
    dvname: string[10];
```

```
    deviceid: integer;
{    message: packed array[1..40] of char;}
    message: string;
    data: packed array[1..200] of char;

    i: integer;


  procedure errorg;
  {common point error get to}
  begin
    writeln('gpib error');
    halt;
  end;

  procedure opengpib;
  {open the gpib driver}
  begin
    writeln('attempting to open gpib');
    err := opendriver('.GPIB driver', refnum);
    if refnum > 0 then
      errorg;
    if err <> noerr then
      begin
        theItem := DoMisc('Unable to open GBIP board driver');
        writeln('cant open gpib driver');
        errorg;
        DoCloseVideo;
      end;
    gpibpointer := @gpib; {assign pointer to the parameter block}
    gpibpointer^.statusptr := @status; {assign pointer to status block}
    {now open the board}
    bdname := gpibslotname;  {get slot number which is part of the name}
{bdname[length(bdname) + 1] := chr(0);}
 {put a null byte at the end, make it C string}
    bdname := concat(bdname, chr(0));
    gpibpointer^.iobuf := @bdname[1]; {point to the board name}
    err := control(refnum, ibfind, ptr(@gpibpointer)); {find the board}
    if err <> noerr then
      begin
        writeln('error finding the gpib board');
        errorg;
        DoCloseVideo;
      end;
    boardid := gpibpointer^.id; {save the board id}
    if boardid < 0 then
      begin
        writeln('board name was not found');
        errorg;
        DoCloseVideo;
      end;
    {now send an interface clear (IFC) }
    {IFC is issued by doing an ibSIC command}
    err := control(refnum, ibsic, ptr(@gpibpointer));
    if err <> noerr then
      begin
        writeln('error sending ibsic command:', err);
        errorg;
      end;
    if gpibpointer^.statusptr^.ibsta < 0 then
      begin
        writeln('status error after sending IFC');
        errorg;
```

```
      end;
    {open the klinger device}
    dvname := klinger;
  {dvname[length(dvname) + 1] := chr(0); }
  {make a C string}
    dvname := concat(dvname, chr(0));
    gpibpointer^.iobuf := @dvname[1];
    err := control(refnum, ibfind, ptr(@gpibpointer));
    if err <> noerr then
      begin
        writeln('device ibfind not successful ', err);
        errorg;
      end;
    deviceid := gpibpointer^.id;
    if deviceid < 0 then
      begin
        writeln('device id not found in driver table');
        errorg;
      end;
    writeln('opened gpib');
  end; {opengpib}

  procedure writegpib (s: string);
  {write string to gpib}
  begin
    message := concat(s, chr(13));
    gpibpointer^.iobuf := @message[1]; {iobuf pionts to the message}
    gpibpointer^.iocount := length(message);
    err := control(refnum, ibWRT, ptr(@gpibpointer));
    if err <> noerr then
      writeln('error writing string to Klinger Controller:', err);
  end; {writegpib}

  procedure selectmo (num: integer);
  {position microscope objective wheel at the selected MO}
  {position X-Y stages for the selected MO}
  {MO's are numbered 1 to 12}
    const
      delta = (36000 div 12); {wheel steps/number of MOs}
      wcorr = 0;  {correction factors to axes if instr gets bumped}
      xcorr = 0;
      ycorr = 0;
    var
      s: string;
      x, y, w: integer;
  begin

{    DoGoHome;}


    wpos[3] := 5929; { 5x ND0 }
    wpos[2] := 2929; { 5x ND1 }
    wpos[1] := -76;  { 5x ND2 }
    wpos[6] := 15000; { 5x ND3 }
    wpos[9] := 24000;{ 10x ND0 }
    wpos[8] := 21000;{ 10x ND1 }
    wpos[7] := 18000;{ 10x ND2 }
    wpos[5] := 12000;{ Mirror }
    wpos[4] := 9000;{ Aperture }

    xpos[3] := -575; {mo number [] x position }
    xpos[2] := -575;
    xpos[1] := -575;
```

```
      xpos[6] := -575;
      xpos[9] := -750;
      xpos[8] := -750;
      xpos[7] := -750;
      xpos[5] := 0;
      xpos[4] := 0;

      ypos[3] := 496; {mo number [] y position }
      ypos[2] := 496;
      ypos[1] := 496;
      ypos[6] := 496;
      ypos[9] := 9906;
      ypos[8] := 9906;
      ypos[7] := 9906;
      ypos[5] := 496;
      ypos[4] := 0;

     {Always position wheel with the Y-Axis run to the negative end of travel}
      writegpib('PY-10650'); {backwards}
      writegpib('RW4000'); { steps }
      writegpib('+W');     { select positive W direction }

      writegpib('OW');     {should already be at the origin from the gohome code}
      writegpib('AW');     { should already be reset from the gohome code}

      w := wpos[num]; {get w position}
      if (w < 30000) or (w >= 0) then
        begin
          writegpib(concat('PW', stringof(wpos[num] + wcorr)));
        end;

      y := ypos[num]; {get y position}
      if y <> 0 then
        begin
{ first position focus ( y ) }
          writegpib('OY'); {The gohome code sets the reference and we return to it}
          writegpib(concat('PY', stringof(abs(y) + ycorr)));
        end;

      x := xpos[num]; {get position}
      if x <> 0 then
        begin
{now position x axis}
          writegpib('OX'); {The gohome code sets the reference and we return to it}
          writegpib(concat('PX', stringof(x + xcorr)));
        end;

    end;


    procedure DoGoHome;
{ gpib stuff for center stages }
    begin

      writeln('*************************************************');
      writeln('Centering Stages to the midpoint of their travel');
      writeln('*************************************************');

      writegpib('RW4000');      { steps/sec }
      writegpib('RY2000');      { steps/sec }
      writegpib('RX2000');      { steps/sec }

      writegpib('-X');     { negative direction }
```

```
    writegpib('NX21500');{  more than the total number of steps }
    writegpib('MX');      { move stage }

    writegpib('-Y');      { negative direction }
    writegpib('NY21500');{ more than the total numbernumber of steps }
    writegpib('MY');      { move stage }

    writegpib('OW');      { MO wheel to origin }
    writegpib('AW');      { resets all displays to o.ooo }

    writegpib('+Y');      { positive direction }
    writegpib('NY10650');{ exactly half the total number of steps }
    writegpib('MY');      { move stage }
    writegpib('AY');      { resets all displays to o.ooo }

    writegpib('NX10650');{number of steps }
    writegpib('+X');{positive direction }
    writegpib('MX'); {move stage }
    writegpib('AX');      { resets all displays to o.ooo }


{ Now enter data from Calibrate Dialog Box }

    writegpib(concat('PW', WDefault));  { FudgeFactor Z }
    writegpib(concat('PX', XDefault));  { FudgeFactor X}
    writegpib(concat('PY', YDefault));  { FudgeFactor Y}

    writegpib('AW');      { resets all displays to o.ooo }
    writegpib('AX');      { resets all displays to o.ooo }
    writegpib('AY');      { resets all displays to o.ooo }

  end;

end.
```

THIS PAGE INTENTIONALLY LEFT BLANK

```
unit DPHistogram;
{  7/29/91 fixto getting and displaying histogram}
{ 08/05/91 changed info line stuff}

interface

  uses
    Picker, DPGlobals, instrintf;

  procedure DoDrawGrid;
  procedure DoPlotData;
  procedure DoColorPicker;
  procedure DoCreateData;
  procedure DoDrawInfoBar;

implementation

  procedure DoColorPicker;
{ brings up Color Picker Dialog box }
    var
      where: Point;
      prompt: str255;
      result: Boolean;
  begin
    where.v := 0;
    where.h := 0;
    prompt := 'Select a histogram color:';
    result := GetColor(where, prompt, plotColor, plotColor);
  end;

  procedure DoDrawGrid;
{procedure DoDrawGrid draw plot grid only }

    const
      vertgrid = 24; {vertical grid spacing}
      horzgrid = 10; {horizontal grid spacing}


    var
      incrementAmt: LongInt;
      i, j: Integer;
      maxValue: LongInt;
      tempString: Str255;
      tempNumber: LongInt;
      width: Integer;      { width of string }
      haxisRect: Rect;  { horizontal axis text location }
      vaxisRect: Rect;  { vertical axis text location }
      numhorzgrid: integer; {number of hor grid lines}
      numvertgrid: integer; { vert grid lines}

  begin
    SetRect(plotFrame, 50, 30, 434, 350);   { name,left,top,right,bottom }
    SetRect(haxisRect, 40, 360, 444, 375);
    PenSize(1, 1);
    with plotFrame do
      begin
        SetRect(vaxisRect, left - 48, top - 5, left - 1, bottom + 5);
        numhorzgrid := (bottom - top) div horzgrid; {figure number of lines we will ha
        numvertgrid := (right - left) div vertgrid;
      end;
    RGBForeColor(gridColor);
    OffsetRect(plotFrame, 4, 4);
    PaintRect(plotFrame);      { shadow }
```

```
    OffsetRect(plotFrame, -4, -4);
    RGBForeColor(whiteRGB);
    PaintRect(plotFrame);        { erase existing plot }
    PaintRect(haxisRect);      { erase existing text }
    PaintRect(vaxisRect);      { erase existing text }
    RGBForeColor(gridColor);
    with plotFrame do
      begin
        for i := 1 to numhorzgrid do
          begin
            j := i * horzgrid + top;
            MoveTo(left, j);  { Horizontal gridlines  (h,v) }
            LineTo(right, j);
          end;
        for i := 1 to numvertgrid do
          begin
            j := i * vertgrid + left;
            MoveTo(j, top);   { Vertical gridlines (h,v)  }
            LineTo(j, bottom);
          end;
        TextFace([]);     { plain text }
        TextFont(Geneva);
        TextSize(9);
        RGBForeColor(blackRGB);
        for i := 0 to numvertgrid do { x axis labels }
          begin
            NumToString((i * 8), tempString);
            width := StringWidth(tempString);
            MoveTo(vertgrid * i + left - (width div 2), bottom + 15);      { h,v }
            DrawString(tempString);
          end;
        tempString := 'Bin Number';
        width := StringWidth(tempString);
        MoveTo(left + ((right - left) div 2) - (width div 2), bottom + 30);
        TextFace([bold]);
        DrawString(tempString);
        TextFace([]);
        maxvalue := scanmax(sortdata); { get biggest value }
        with plotFrame do
          incrementAmt := 1 + trunc(maxValue / numhorzgrid);
        for i := 0 to numhorzgrid do   { y axis labels }
          begin
            tempNumber := incrementAmt * i;
            NumToString(tempNumber, tempString);
            width := StringWidth(tempString);      { used to right justify text }
            MoveTo(left - 9 - width, bottom - ((i * horzgrid) - 4));
            DrawString(tempstring);
          end;
        TextFace([bold]);
        width := StringWidth('Counts'); { used to right justify text }
        MoveTo(left - 9 - width, bottom - (i * horzgrid));
        DrawString('Counts');
      end;     { with plotframe do }
    RGBForeColor(blackRGB);
    FrameRect(plotFrame);
    TextFont(0);     { 0 is system font }
    TextFace([]);
    TextSize(12)
  end;

  procedure DoCreateData;

    var
```

```
        temp: LongInt;
        i: Integer;
        maxValue: LongInt;

    begin
      dataSaved := FALSE;  { new data, therefore it ain't saved! }
      acquire(numberofscans);
      for i := 1 to numberofbins do
        dataarray[i] := trunc((sortdata[i - 1]));
    { find max value and normalize to it }
      maxvalue := trunc((scanmax(sortdata))); {get biggest value}
      if maxvalue = 0 then
        writeln('max value = 0 error')
      else
        begin
          with plotFrame do
            begin
              for i := 1 to 128 do
                begin
                  dataArray[i] := round((dataArray[i] / maxValue) * (bottom - top - 1));
                end;
            end;
        end; {maxvalue<>0}
    end;    { of procedure DoCreateData }

    procedure DoPlotData;
  { procedure DoPlotData plots histogram data }

      var
        bar: Rect;
        i: Integer;
        temp: LongInt;
        maxvalue: longint;

    begin
      RGBForeColor(plotColor);  { set color to histogram color }
      for i := 1 to 128 do          { change 128 to numberofBins }
        begin
          with plotFrame do    { for rectangle dimensions }
            begin
              if dataArray[i] >= 1 then
                begin
                  SetRect(bar, left + ((i * 3) - 3), bottom - (dataArray[i]), left + (i *
 bottom);            {left,top,right,bottom}
                  PaintRect(bar);
                end;
            end;
        end;
      RGBForeColor(blackRGB);
      FrameRect(plotFrame);
    end;

    function DoTimeElapsed: longint;
      var
        secs: longint;
        date: DateTimeRec;

    begin
      GetDateTime(secs);
      time := secs - time;
      Secs2Date(time, date);
    end;
```

```
    procedure DivideLine;
  { draws dividing line in info bar }

    begin
      Move(5, 4);
      Line(0, -16);
      Move(5, 12);
    end;

    procedure DoDrawInfoBar;
  { procedure DoDrawInfoBar draws info bar at bottom of window }
      var
        tempRect: Rect;
        tempStr: Str255;
        totalcounts, maxcounts: longint;

      procedure drawstrnum (s: string; num: longint; field: integer);
      begin
        tempStr := stringof(num : field);
        DrawString(Concat(s, tempStr));
        DivideLine;
      end;

    begin
      TextFace([]);    { plain text }
      TextFont(Geneva);
      TextSize(9);
      with myWindow^.portRect do
        begin
          RGBForeColor(gridColor);
          SetRect(tempRect, left, bottom - 16, right, bottom);
          PaintRect(tempRect);
          RGBForeColor(blackRGB);
          MoveTo(left, bottom - 16);
          LineTo(right, bottom - 16);
          MoveTo(left + 3, bottom - 4);
          DrawString(Concat('Object: ', objectName));
          DivideLine;
          drawstrnum('Scans: ', numberofscans, 5);
          drawstrnum('Scans/Sec: ', scanrate[scanspeednum], 2);
          DrawString(Concat('Mag/Filter: ', magFilter));
          DivideLine;
          totalcounts := scantotal(sortdata);
          maxcounts := scanmax(sortdata);
          drawstrnum('Counts: ', totalcounts, 8);
          drawstrnum('Peak: ', maxcounts, 5);
          drawString(concat('PPPS: ', stringof(peakpulserate : 6)));
          DivideLine;
  {if totalcounts <> 0 then}
  {drawString(concat('Q: ', stringof(maxcounts / totalcounts : 8 : 6)));}
        end;
      TextFont(0);     { 0 is system font }
      TextFace([]);    { plain text }
      TextSize(12)
    end;


  end.
```

```
unit DPFile;
{ 8/5/91: Started revision history RS}
{ }

interface

  uses
    DPGlobals, DPDialogs, DPHistogram;

  procedure DoSaveAFile (fName: str255; vRefNum: Integer);
  procedure DoSaveFileName;
  procedure DoOpenAFile (fName: str255; vRefNum: Integer);
  procedure DoGetFileName;
  procedure DoSavePICTFile;
  procedure DoSaveTEXTFile (fName: str255; vRefNum: integer);
  procedure DoSaveTEXTFileName;

implementation

  procedure DoSaveAFile;
{ saves the current data to a file indicated by the user  RS}

    var
      refnum: Integer;
      length: longint;
      stuff: charsHandle;

  begin
    errno := Create(fname, vrefnum, creator, filetype);
    if errno <> noerr then
      begin
        DoError(errno);
        exit(DoSaveAFile);
      end;
    errno := FSOpen(fname, vrefnum, refnum);
    if errno <> noerr then
      begin
        DoError(errno);
        exit(DoSaveAFile);
      end;
    length := 128 * 4;    { 128 longint * 4 bytes per integer }
    errno := FSWrite(refnum, length, ptr(@dataArray));
    if errno <> noErr then
      begin
        DoError(errno);
        exit(DoSaveAFile);
      end;
    SetWTitle(FrontWindow, fileName);
    errno := FSClose(refnum);
    if errno <> noerr then
      begin
        DoError(errno);
        exit(DoSaveAFile);
      end;

  end; { Procedure SaveAFile }

  procedure DoSaveFileName;
{ displays SFPutFile dialog RS }

    var
      where: Point;
      prompt: Str255;
```

```
        reply: SFReply;
        refnum: integer;
        length: longint;
        stuff: charsHandle;

    begin
      where.v := 0;
      where.h := 0;
      prompt := 'Save current document as:';

      SFPutFile(where, prompt, fileName, nil, reply);

      if reply.good then
        begin
          fileName := reply.fName;
          vRefNum := reply.vRefNum;
          DoSaveAFile(reply.fName, reply.vRefNum);
          dataSaved := TRUE;
          hasFileName := TRUE;
        end;
    end;

    procedure DoOpenAFile;
  {  opens a file indicated by the user RS}

      var
        refnum: Integer;
        size: longint;

    begin
      errno := FSOpen(fname, vrefnum, refnum);
      if errno <> noErr then
        begin
          DoError(errno);
          exit(DoOpenAFile);
        end;
      errno := GetEOF(refnum, size);
      if errno <> noErr then
        begin
          DoError(errno);
          exit(DoOpenAFile);
        end;
      if size > sizeof(dataArray) then
        size := sizeof(dataArray);
      errno := FSRead(refnum, size, @dataArray);
      if errno <> noErr then
        begin
          DoError(errno);
          exit(DoOpenAFile);
        end;
    end; { Procedure OpenAFile }

    procedure DoGetFileName;
  { displays SFGetFile dialog to select file RS }
      var
        numTypes: Integer;
        prompt: str255;  { not used }
        reply: SFReply;
        typeList: SFTypeList;
        where: Point;

    begin
      where.v := 0;
```

```
    where.h := 0;
    numTypes := 1;   {get Text types}
    typeList[0] := 'DIMG';    { file type for our application }

    SFGetFile(where, Prompt, nil, numTypes, typeList, nil, reply);
    if reply.good then  { if everything is OK }
      begin
        SetWTitle(FrontWindow, reply.fName);
        fileName := reply.fName;
        vRefNum := reply.vRefNum;
        dataSaved := TRUE;
        hasFileName := TRUE;
        DoOpenAFile(reply.fname, reply.vrefnum);
      end;
  end;          { procedure DoGetFileName  }

  procedure PutPICTData (dataPtr: Ptr; byteCount: Integer);
    var
      longCount: LongInt;
      errno: Integer;
  begin
    longCount := byteCount;
    PICTCount := PICTCount + byteCount;
    errno := FSWrite(globalRef, LongCount, dataPtr);
    if newPictHand <> nil then
      newPICTHand^^.picSize := PICTCount;
  end;

  procedure DoSavePICTFile;
{ saves PICT File to disk; See Inside Macintosh V-90 for code }
    var
      errno: OSErr;
      i: Integer;
      where: Point;
      longCount: LongInt;
      longZero: LongInt;
      pFrame: Rect;
      reply: SFReply;
      myProcs: QDProcs;

  begin
    where.h := 0;
    where.v := 0;
    SFPutFile(where, 'Save PICT file as:', Concat(fileName, '.PICT'), nil, reply);
    if reply.good then
      begin
        errno := Create(reply.fName, reply.vRefNum, '????', 'PICT');
        if (errno = noErr) or (errno = dupfnerr) then
          begin
            errno := FSOpen(reply.fName, reply.vRefNum, globalRef);
            if errno <> noErr then
              DoError(errno);
            SetStdProcs(myProcs);
            myWindow^.grafProcs := @myProcs;
            myProcs.putPicProc := @putPictData;
            longZero := 0;
            longCount := 4;
            PICTCount := SizeOf(picture);
            for i := 1 to 512 div 4 + SizeOf(picture) do
              errno := FSWrite(globalRef, longCount, @longZero);
            with plotFrame do
              SetRect(pFrame, left, top, right, bottom);
            newPICTHand := nil;
```

```
            newPICTHand := OpenPicture(pFrame);
            DoDrawGrid;
            DoPlotData;
            ClosePicture;
            errno := SetFPos(globalRef, fsFromStart, 512);
            longCount := SizeOf(Picture);
            errno := FSWrite(globalRef, longCount, Ptr(newPICTHand^));
            if errno <> noErr then
              DoError(errno);
            errno := FSClose(globalRef);
            if errno <> noErr then
              DoError(errno);
            myWindow^.grafProcs := nil;
            KillPicture(newPICTHand);
          end
        else
          DoError(errno);
      end; { if reply.good }
  end;{ procedure DoSavePICTFile }

  procedure DoSaveTEXTFile;
{ Saves unnormalized, sorted data to disk as a }
{ Microsoft Excel TEXT document RS }

    var
      counts: Str255;
      errno: OSErr;
      i: Integer;
      strLength: LongInt;
      tempStr: Str255;
      systemTime: Str255;
      date: Str255;
      secs: LongInt;

    const
      fileCreator = 'XCEL';    { Excel creator type }

    procedure writeText (writeText: Str255);
{ write text followed by a carriage return RS }

    begin
      writeText := Concat(writeText, CHR(13));     { CHR(13) is carriage return }
      strLength := length(writeText);
      errno := FSWrite(globalRef, strLength, @writeText[1]);      { first byte is lengt
      if errno <> noErr then
        DoError(errno);
    end;  { procedure writeText }

  begin
    errno := Create(fName, vRefNum, fileCreator, 'TEXT');
    if (errno = noErr) or (errno = dupfnerr) then
      begin
        errno := FSOpen(fName, vRefNum, globalRef);
        if errno <> noErr then
          DoError(errno);

        GetDateTime(secs);
        IUDateString(secs, longDate, date);
        IUTimeString(secs, true, SystemTime);
        writeText(Concat('Date: ', Chr(9), date));
        writeText(Concat('Time: ', Chr(9), SystemTime));
        writeText(Concat('Object Name: ', Chr(9), objectName));
        writeText(Concat('Observatory: ', Chr(9), observatory));
```

```
        writeText(Concat('Right Ascension: ', Chr(9), RAValue));
        writeText(Concat('Declination: ', Chr(9), decValue));
        writeText(Concat('Scan Speed: ', Chr(9), scanSpeedValue));
        NumToString(numberofScans, tempStr);
        writeText(Concat('Number of Scans: ', Chr(9), tempStr));
        writeText(Concat('Bin Number    ', Chr(9), 'Counts'));
        for i := 0 to integer(numberofBins) do
          begin
            NumToString(sortdata[i], counts);
            NumToString(i, tempStr);
            writeText(Concat(tempStr, Chr(9), counts));
          end;
        errno := FSClose(globalRef);
        if errno <> noErr then
          DoError(errno);
      end;
  end;{ procedure DoSaveTEXTFile }

  procedure DoSaveTEXTFileName;
    var
      where: Point;
      reply: SFReply;

  begin
    where.h := 0;
    where.v := 0;
    SFPutFile(where, 'Export TEXT file as:', Concat(objectName, '.TEXT'), nil, reply);
    if reply.good then
      DoSaveTEXTFile(reply.fName, reply.vRefNum);
  end;

end.
```

THIS PAGE INTENTIONALLY LEFT BLANK

```
unit DPPrint;
{ By Robert Slavey }
{July 25, 1991: Original code }
{ printing code; should be in separate segment }
{ PrintStuff from Macintosh Technical Note #161}

interface
  uses
    PrintTraps, DPHistogram;

  procedure PrintStuff;

implementation

  function DetermineNumberOfPagesInDoc (rPage: Rect): Integer;
{ for DigitalProfiler, number of pages is always one }
  begin
    DetermineNumberofPagesInDoc := 1;
  end;

  procedure CheckMyDialogButton;
  begin
  end;

  procedure DrawStuff (rpage: Rect; thePrPort: GrafPtr; pageNumber: Integer);
{ call drawing routines here }
    var
      temp: Rect;

  begin
    SetPort(thePrPort);
    DoDrawGrid;
    DoPlotData;
  end;

  procedure PostPrintingErrors (theError: Integer);
  begin
  end;

  procedure PrintStuff;
    var
      copies, firstpage, lastPage, loop, numberOfCopies, pageNumber, printmgrsResFile,
 realNumberofPagesinDoc: Integer;
      PrintError: longInt;
      oldPort: GrafPtr;
      thePrRecHdl: THPrint;
      thePrPort: TPPrPort;
      theStatus: TPrStatus;
      PrintingStatusDialog: DialogPtr;
      theDialog: DialogPtr;
      theItem: Integer;

  begin
    GetPort(oldPort);
    thePRRecHdl := THPrint(newHandle(SIZEOF(TPrint)));
{    UnLoadTheWorld; find out where this is defined }
    if (MemError = noerr) and (thePRRecHdl <> nil) then
      begin
        PROpen;
        if (PRError = NoErr) then
          begin
            printmgrsResFile := CUrResFile;
            PrintDefault(thePrRecHdl);
```

```
            if (PRError = noErr) then
              begin
                if (PrStlDialog(thePrRecHdl)) then
                  begin
                    realNumberofPagesInDoc :=
DetermineNumberOfPagesInDoc(thePrRecHdl^^.PrInfo.rPage);

                  if (prJobDialog(thePrRecHdl)) then
                    begin
                      theDialog := GetNewDialog(128, nil, Pointer(-1));    { display
cancel  printing dialog }
                    numberOfCopies := thePRRecHdl^^.prJob.icopies;
                    firstPage := thePRRecHdl^^.prJob.iFstPage;
                    lastPage := thePRRecHdl^^.prJob.iLstPage;
                    thePRRecHdl^^.prJob.ifstPage := 1;
                    thePRRecHdl^^.prJob.iLstPage := 9999;
                    if (realNumberofPagesinDoc < lastPage) then
                      lastpage := realNumberofPagesinDoc;
                    for copies := 1 to numberOfCopies do
                      begin
                        PrintingStatusDialog := GetNewDialog(257, nil, pointer(-1))
                        thePRRecHdl^^.prJob.pIdleProc := @CheckMyDialogButton;
                        UseResFile(printmgrsResFile);
                        thePrPort := PrOpenDoc(thePrRecHdl, nil, nil);
                        if (PrError = noErr) then
                          pageNumber := firstPage;
                        while ((PageNumber <= lastPage) and (prError = noErr)) do
                          begin
                            PrOpenPage(thePrPort, nil);
                            if (PrError = noErr) then
                              begin
                                DrawStuff(thePrRecHdl^^.prInfo.rpage,
GrafPtr(thePrPort), pageNumber);
                              end;
                            PrClosepage(thePrPort);
                            pageNumber := pageNumber + 1;
                          end;
                      end;
                    PrCloseDoc(thePrPort);
                  end;          { of copies loop }
                end
              else
                PrSetError(iPrAbort); { cancel from the job dialog }
            end
          else
            PrSetError(iPrAbort);      { cancel from the style dialog }
        end;
    end;
  if (thePRRecHdl^^.prJob.bJDocLoop = bSpoolLoop) and (prError = noErr) then
    PrPicFile(thePrRecHdl, nil, nil, nil, thestatus);
  PrintError := PrError;
  PRClose;
  if (printerror <> NoErr) then
    PostPrintingErrors(printError);

  if (thePrRecHdl <> nil) then
    DisposHandle(Handle(thePrRecHdl));
  if (printingStatusDialog <> nil) then
    DisposDialog(PrintingStatusDialog);
  DisposDialog(theDialog);                { get rid of dialog box          }
  SetPort(oldPort);
end;
```

```
end.
```

THIS PAGE INTENTIONALLY LEFT BLANK

```
unit DPVideo;
{ 08/05/91  : Displayed video on screen}
{ 08/06/91  : Copy video image to Clipboard RS }

interface
  uses
    Traps, Slots, SANE, QuickDraw, DVCDriver, DPGlobals, DPDialogs, DPHistogram;

  procedure DoInitVideo;
  procedure DoDrawVideo;
  procedure DoCloseVideo;
  procedure DoVideoToClipboard;

implementation

  const
    _QD32Trap = $AB03;

  var
    err: OSErr;
    gBestImage, gInBackground: BOOLEAN;
    gScreenSize, gCardSlot: INTEGER;
    gBrightness, gContrast, gTint, gColor, gPicture: INTEGER;
    gBounds: Rect;
    gOffBounds: array[smallTV..largeTV] of Rect;
    gMac: SysEnvRec;
    gInfoRec: MTVInfoRecord;
    temp: Str255;
    num: LongInt;

    myRefNum: integer;

  procedure DoCloseVideo;
{ closes MTV board driver }
  begin
    if gRefNum <> 0 then
      err := CloseDriver(gRefNum);
  end;

  procedure DoInitVideo;
{ initializes MTV card; refer to MinimalPascal source from AAPS }

  begin
    gBestImage := FALSE;
    gInBackground := FALSE;
    gScreenSize := smallTV;
    gCardSlot := 0;
    gRefNum := 0;
    gBrightness := 30;
    gContrast := 60;
    gTint := 50;
    gColor := 75;
    gPicture := 100;
    with mywindow^.portrect do
      begin
        SetRect(gOffBounds[smallTV], right - 128 - 25, 30, right - 25, 30 + 108);
      end;
    SetRect(gOffBounds[smallTV], 0, 11, 128, 119);{card only supports small TV}

    SetRect(gOffBounds[largeTV], 0, 22, 256, 238); {not using large TV}
    gBounds := gOffBounds[gScreenSize];
    OffsetRect(gBounds, 479, 30 - 11);   { assign location of picture }
    gCardSlot := MTVFirstAvailableCard;
```

```
    if gCardSlot = 0 then
      begin
        theItem := DoMisc('Unable to find a MicroTV card that is not already in use.');
        exit(DoInitVideo);
      end;
    err := MTVGetInfo(gRefNum, gInfoRec);
    err := MTVOpenDriver(gCardSlot, gRefNum);
    writeln('MTV Card ref num: ', gRefNum, 'error #', err);
    if err <> noErr then
      begin
        NumToString(Num2Longint(err), temp);
        DoError(err);
        theItem := DoMisc(Concat('Unable to open MicroTV driver. Error ID #', temp));
        exit(DoInitVideo);
      end;
    if MTVGetInfo(gRefNum, gInfoRec) <> noErr then
      begin
        theItem := DoMisc('Unable to obtain MTV card info.');
        DoCloseVideo;
        exit(DoInitVideo);
      end;
    err := MTVDoubleBuffer(gRefNum, swDoubleBuffer);
    { ignore errors for now }
    if err <> Err then
      begin
        NumToString(Num2Longint(err), temp);
        theItem := DoMisc(Concat('MTV Double buffer error. Error ID #', temp));
        DoCloseVideo;
        exit(DoInitVideo);
      end;
    err := MTVDisplay(gRefNum, smallTV + microTVGray, FALSE);
    { ignore errors for now }
    if err <> Err then
      begin
        NumToString(Num2Longint(err), temp);
        theItem := DoMisc(Concat('MTV display error. Error ID #', temp));
        DoCloseVideo;
        exit(DoInitVideo);
      end;
    err := MTVPower(gRefNum, TRUE);
    {}
    if err <> Err then
      begin
        NumToString(Num2Longint(err), temp);
        theItem := DoMisc(Concat('MTV power on error. Error ID #', temp));
        exit(DoInitVideo);
      end;
  end;

  procedure DrawWindow;
{ uses CopyBits to copy image to main screen RS}
    var
      usePortColors: Boolean;

  begin
    usePortColors := TRUE;
    err := MTVCopyPict(gRefNum, CWindowPtr(myWindow), gOffBounds[gScreenSize], gBounds,
 srcCopy, TRUE, usePortColors);
  end;

  procedure DoDrawVideo;
{ if there is something to show, show it RS}
```

```
  begin
    if MTVFrameBuilt(gRefNum) then
      DrawWindow;
  end;

  procedure DoVideoToClipboard;
{ will copy one frame of video image to the clipboard RS }
    var
      errno: Longint;          { error code returned by functions }
      datalength: Longint;     { length of data }
      frame: Rect;             { box for open picture }
      image: PicHandle;        { video frame to be copied }
      I: integer;

  begin
    errno := ZeroScrap;  { clear existing Clipboard contents }
    if errno <> noErr then
      DoError(errno);
    SetRect(frame, 0, 0, 108, 128);
    image := nil;
    image := OpenPicture(frame);    { all drawing calls until ClosePicture will be copie
    DoDrawGrid;
    DoPlotData;
    ClosePicture;
    dataLength := SizeOf(image);
    errno := PutScrap(dataLength, 'PICT', ptr(image^));
    if errno <> noErr then
      DoError(errno);
    KillPicture(image);
  end;


end.
```

THIS PAGE INTENTIONALLY LEFT BLANK

```
unit DPUtilities;
{ handles menu commands and various controls }
{ by Robert Slavey }
{ July 25,1991: Original Code  RS }
{ Aug 20, 1991: Added range checking to DoEnableChannel RS }
{ Aug 22, 1991: Added DoTextWindow routine RS }
{ 8/23/91  changed parameter box to display digital thresh instead of discr thr}
{          calls initdiscriminators() when threshold is changed}
{ 8/29/91 added DoFindStar and DoRealTimeOptions procedures RS }
{ 8/30/91 added Calibration options procedures EM }
{ 9/2/91 removed DoFindStar and disabled DoRealTimeOptions EM}
interface

  uses

    SANE, DPGlobals, DPDialogs, instrintf, sercomm, DPGPIB, DPFile, DPHistogram, DPPrin
 DPVideo;

  procedure DoOpenMenu;
  function DoSaveChanges (reason: Str255): Integer;
  procedure DoPrinting;
  procedure DoRealTime;
  procedure DoRealTimeOptions;
  procedure DoNumberofScans;
  procedure DoNumberofBins;
  procedure DoNumberofBoards;
  procedure DoThreshold;
  procedure DoEnableChannel;
  procedure DoScanSpeed (theItem: Integer);
  procedure DoMOMenu (theItem: Integer);
  procedure DoResolution;
  procedure DoCalibrateStages;
  procedure DoParameters;
  procedure DoAcquireData;
  procedure DoClearData;
  procedure DoImageZoomIn;
  procedure DoImageZoomout;
  procedure DoCenterStages;
  procedure DoScrollImage (whichControl: ControlHandle; part: integer; myPt: Point);
  procedure DoObjectInfo;
  procedure DoMenuDeskew;
  procedure DoGPIBCommand;
  procedure DoSerialCommand;
  procedure DoDrawStatusWindow;
  procedure DoEDStatus;
  procedure DoTextWindow;

implementation

  procedure DoOpenMenu;

  begin
    DoGetFilename;    { open a file }
    DoDrawGrid;
    DoPlotData;      { and plot it }
  end;

  function DoSaveChanges;
{ ••• Procedure DoSaveChanges displays SaveChanges... dialog box ••• }
    var
      savePort: GrafPtr;
      theItem: Integer;
      theDialog: DialogPtr;
```

```
begin
  if not dataSaved then
    begin
      GetPort(savePort);
      SetCursor(arrow);
      ParamText('', Concat('"', fileName, '"'), reason, '');
      theDialog := GetNewDialog(QuitID, nil, Pointer(-1));   { get dialog box }
      DoCenterDialog(theDialog);
      DoOutline(theDialog);
      ModalDialog(nil, theItem);              { put dialog box up; get result  }
      case theItem of
        1:   { Save button}
          begin
            if hasFileName then
              DoSaveAFile(fileName, vRefNum);
            if not hasFileName then
              DoSaveFileName;
            dataSaved := TRUE;
            SetWTitle(myWindow, fileName);
          end;
        2:   { Cancel button }
          begin
          end;
        3:   { Don't Save button }
          begin
          end;
      end;
      DisposDialog(theDialog);              { get rid of dialog box          }
      SetPort(savePort);
    end;
  DoSaveChanges := theItem;
end; { of DoSaveChanges }

procedure DoPrinting;
begin
  PrintStuff;
end;

procedure DoRealTime;
{go get a small number of scans, display, and save (if selected) }
  var
    tempnumscans: longint;
    imagestr: Str255;

begin
  tempnumscans := NumberOfScans; { save the user selected value}
  numberOfScans := numberofscans; { use current value (used to be 10) }
  DoAcquireData;
  NumToString(imageNumber, imagestr); { current image number }
  if saverealtimeflag then
    begin
      DoSaveTEXTFile(Concat(realtimefilename, ' ', imagestr), realtimerefnum);
    end;
  numberOfScans := tempnumscans; {restore user value}
  imageNumber := imageNumber + 1;
end;

procedure DoRealTimeOptions;
  var
    where: Point;
    prompt: Str255;
    reply: SFReply;
```

```
  begin
    realtimeflag := TRUE;
    saverealtimeflag := FALSE;
    theItem := DoMisc('Save the data for each image during real time acquisition?');
    if theItem = 1 then
      begin
        where.v := 0;     { position of SFPutFile dialog }
        where.h := 0;
        prompt := 'Save realtime data as:';
        SFPutFile(where, prompt, objectName, nil, reply);
        if reply.good then
          begin
            realtimefileName := reply.fName;
            realtimerefnum := reply.vRefNum;
          end;
        saverealtimeflag := TRUE;
      end;
    imageNumber := 1;     { start with image number one }
  end;

  procedure DoCalibrateStages;
{ procedure DoCalibrateStages brings up dialog to calibrate stage positions }

    var
      box: Rect;
      item: Handle;
      itemType: Integer;
      result: Integer;
      savePort: GrafPtr;
      theDialog: DialogPtr;
      theItem: Integer;
      theControl: ControlHandle;
      value: Integer;

  begin
    GetPort(savePort);
    theDialog := GetNewDialog(CalibrateID, nil, Pointer(-1)); { get dialog box }
    DoCenterDialog(theDialog);
    DoOutline(theDialog);
    GetDItem(theDialog, 4, itemType, item, box);
    SetIText(item, WDefault);
    SelIText(theDialog, 4, 0, 32767);

    GetDItem(theDialog, 5, itemType, item, box);
    SetIText(item, XDefault);

    GetDItem(theDialog, 6, itemType, item, box);
    SetIText(item, YDefault);
    repeat
      begin
        ModalDialog(nil, theItem);              { put dialog box up; get result  }
        case theItem of
          1:    { OK Button }
            begin
              GetDItem(theDialog, 4, itemType, item, box);
              GetIText(item, WDefault);

              GetDItem(theDialog, 5, itemType, item, box);
              GetIText(item, XDefault);

              GetDItem(theDialog, 6, itemType, item, box);
              GetIText(item, YDefault);
```

```
            end;

          3:   { Revert button }
            begin
              result := DoMisc('Revert to default position values?');
              DoOutline(theDialog);
              if result = 1 then  { OK Button }
                begin
                  WDefault := '0';
                  XDefault := '0';
                  YDefault := '0';

                  GetDItem(theDialog, 4, itemType, item, box);
                  SetIText(item, WDefault);
                  SelIText(theDialog, 4, 0, 32767);
                  GetDItem(theDialog, 5, itemType, item, box);
                  SetIText(item, XDefault);

                  GetDItem(theDialog, 6, itemType, item, box);
                  SetIText(item, YDefault);
                end;
            end;
        end;
      end;
    until (theItem = 1) | (theItem = 2);     { OK or Cancel button }
    DisposDialog(theDialog);              { get rid of dialog box          }
    SetPort(savePort);
  end; { Procedure DoCalibrateStages}



  procedure DoResolution;
{ procedure DoResolution brings up dialog to set stage resolutions }

    var
      box: Rect;
      findText: Str255;
      item: Handle;
      itemType: Integer;
      result: Integer;
      savePort: GrafPtr;
      theDialog: DialogPtr;
      theItem: Integer;

  begin
    GetPort(savePort);
    theDialog := GetNewDialog(ResolutionID, nil, Pointer(-1)); { get dialog box }
    DoCenterDialog(theDialog);
    DoOutline(theDialog);
    GetDItem(theDialog, 4, itemType, item, box);
    SetIText(item, moRes);
    SelIText(theDialog, 4, 0, 32767);

    GetDItem(theDialog, 5, itemType, item, box);
    SetIText(item, horizRes);

    GetDItem(theDialog, 6, itemType, item, box);
    SetIText(item, focusRes);

    GetDItem(theDialog, 7, itemType, item, box);
    SetIText(item, irRes);
    repeat
```

```
      begin
        ModalDialog(nil, theItem);              { put dialog box up; get result  }
        case theItem of
          1:   { OK Button }
            begin
              GetDItem(theDialog, 4, itemType, item, box);
              GetIText(item, moRes);

              GetDItem(theDialog, 5, itemType, item, box);
              GetIText(item, horizRes);

              GetDItem(theDialog, 6, itemType, item, box);
              GetIText(item, focusRes);

              GetDItem(theDialog, 7, itemType, item, box);
              GetIText(item, irRes);
            end;
          3:   { Revert button }
            begin
              result := DoMisc('Revert to default resolution values?');
              if result = 1 then  { OK Button }
                begin
                  moRes := moResDefault;
                  horizRes := horizResDefault;
                  focusRes := focusResDefault;
                  irRes := irResDefault;
                  GetDItem(theDialog, 4, itemType, item, box);
                  SetIText(item, moRes);
                  SelIText(theDialog, 4, 0, 32767);
                  GetDItem(theDialog, 5, itemType, item, box);
                  SetIText(item, horizRes);

                  GetDItem(theDialog, 6, itemType, item, box);
                  SetIText(item, focusRes);

                  GetDItem(theDialog, 7, itemType, item, box);
                  SetIText(item, irRes);
                end;
              DoOutline(theDialog);
            end;
        end;   { case }
      end;
    until (theItem = 1) | (theItem = 2);     { OK or Cancel button }
    DisposDialog(theDialog);              { get rid of dialog box          }
    SetPort(savePort);
  end; { Procedure DoResolution}


  function DoGetValue (previousValue: LongInt): LongInt;
    var
      box: Rect;
      item: Handle;
      itemType: Integer;
      numberStr: Str255;
      savePort: GrafPtr;
      theDialog: DialogPtr;
      theItem: Integer;
      theNumber: Integer;

  begin
    GetPort(savePort);
    theDialog := GetNewDialog(NumberID, nil, Pointer(-1)); { get dialog box }
    DoCenterDialog(theDialog);
```

```
    DoOutline(theDialog);
    GetDItem(theDialog, 3, itemType, item, box);
    NumToString(previousValue, numberStr);
    SetIText(item, numberStr);      { set text to previous value }
    SelIText(theDialog, 3, 0, 32767);
    ModalDialog(nil, theItem);            { put dialog box up; get result  }
    GetDItem(theDialog, 3, itemType, item, box);
    GetIText(item, numberStr);      { get new value }
    DisposDialog(theDialog);              { get rid of dialog box          }
    SetPort(savePort);
    case theItem of
      1:  { OK button }
        StringToNum(numberStr, DoGetValue);
      2:  { Cancel button }
        DoGetValue := previousValue;     { keep existing value }
    end; { procedure DoGetValue }
  end;

  procedure DoMoMenu;
{ moves MO wheel to selected MO }
    var
      i: Integer;

  begin
    SetCursor(watch);
    for i := 1 to CountMItems(GetMHandle(MOMenu)) do
      CheckItem(GetMHandle(MOMenu), i, FALSE);     { uncheck all menu items }
    monum := mopos[theItem]; {map menu sel to MO number 1 to 12}
    selectmo(monum);  {tell the wheel to turn to selected MO}
    CheckItem(GetMHandle(MOMenu), theItem, True); { check mark by selected MO }
    GetItem(GetMHandle(MOMenu), theItem, magFilter);
    DoDrawInfoBar;
    SetCursor(arrow);
  end;

  procedure DoNumberofScans;

  begin
    ParamText('Number of scans:', '', '', '');
    numberofScans := DoGetValue(numberofScans);
    DoDrawInfoBar;
  end;

  procedure DoNumberofBins;

  begin
    ParamText('Number of bins:', '', '', '');
    numberofBins := DoGetValue(numberofBins);
  end;

  procedure DoNumberofBoards;

  begin
    ParamText('Number of boards:', '', '', '');
    numberofboards := numboards + 1; {numboards is 0 to 7, display 1 to 8}
    numberofboards := DoGetValue(numberofboards);
    if numberofboards > 8 then
      numberofboards := 8;
    if numberofboards < 1 then
      numberofboards := 8;
    numboards := integer(numberofboards) - 1;
  end;
```

```
procedure DoThreshold;

begin
  ParamText('Discriminator threshold:', '', '', '');
  threshold := DoGetValue(threshold);
  initdiscriminators(threshold); {set up hardware}
end;

procedure DoEnableChannel;

  var
    box: Rect;
    item: Handle;
    itemType: Integer;
    channelStr: Str255;
    channelNum: LongInt;
    savePort: GrafPtr;
    theDialog: DialogPtr;
    theItem: Integer;
    theNumber: Integer;

begin
  GetPort(savePort);
  theDialog := GetNewDialog(EnableID, nil, Pointer(-1)); { get dialog box }
  DoCenterDialog(theDialog);
  DoOutline(theDialog);
  SelIText(theDialog, 6, 0, 32767);    { item 6 is edit text for channel }
  repeat
    ModalDialog(nil, theItem);              { put dialog box up; get result  }
    GetDItem(theDialog, 6, itemType, item, box);      { item 6 is edit text for chann
    GetIText(item, channelStr);   { get value for channel }
    StringToNum(channelStr, channelNum);   { convert to longInt }
    if channelNum > 64 then
      begin
        theNumber := DoMisc('Channel value must be between 1 and 64.');
        DoOutline(theDialog);
        if theItem <> 2 then
          theItem := 0;
      end;
    case theItem of
      2:     { enable button }
        begin
          enablechannel(integer(channelNum));
        end;
      3:     { disable button }
        begin
          disablechannel(integer(channelNum));
        end;
      4:     { disable all button }
        begin
          disableallchannels;
        end;
      5:     { enable all button }
        begin
          enableallchannels;
        end;
      8:     { Cancel button }
        begin
          theItem := 1;
        end;
      otherwise
        begin
        end;
```

```
        end;     { case }
        SelIText(theDialog, 6, 0, 32767);
      until theItem = 1;   { done button }
      initallboards(scanspeednum, 128);
      DisposDialog(theDialog);              { get rid of dialog box }
      SetPort(savePort);
    end;    { procedure DoGetValue }


    procedure DoScanSpeed;
  { sets scan mirror speed }
      var
        i: Integer;

    begin
      for i := 1 to CountMItems(GetMHandle(SpeedMenu)) do
        CheckItem(GetMHandle(SpeedMenu), i, FALSE);
      CheckItem(GetMHandle(SpeedMenu), theItem, True);
      GetItem(GetMHandle(SpeedMenu), theItem, scanSpeedValue);
      scanSpeedNum := theItem - 1;
      DoDrawInfoBar;
      initallboards(scanspeednum, 128);
      setmotorspeed(scanspeednum);
   {initEverything;}
    end;

    procedure DoParameters;
  { gives some current paramaters RS }

      var
        savePort: GrafPtr;
        tempRect: Rect;
        theDialog: DialogPtr;
        theItem: Integer;
        param1: string;
        threshstr: Str255;

    begin
      GetPort(savePort);
      SetCursor(Arrow);
      ShowCursor;
      param1 := stringof(scanrate[scanspeednum]);
      NumToString(threshold, threshstr);
      ParamText(param1, stringof(numberofbins), stringof(whatissrlength),
   stringof(whatisdthresh));
      theDialog := GetNewDialog(ParameterID, nil, Pointer(-1));    { get dialog box  }
      DoCenterDialog(theDialog);
      DoOutline(theDialog);
      ModalDialog(nil, theItem);              { put dialog box up; get result  }
      DisposDialog(theDialog);               { get rid of dialog box          }
      SetPort(savePort);
    end; { of DoParamters }

    procedure DoAcquireData;
      var
        item: Integer;

    begin
  {    if not dataSaved then}
  {item := DoSaveChanges ( 'acquiring new data' );    }
  {    if item <> 2 then    }
  { 2 is cancel button }
      begin
```

```
      SetCCursor(coffee);
      DoCreateData;
      DoDrawGrid;
      DoPlotData;
      DoDrawInfoBar;
      SetCursor(arrow);
    end;
  end;

  procedure DoClearData;
    var
      i: integer;
  begin
    for i := 0 to numberofbins do { fill sort dat with zeros }
      sortdata[i] := 0;
    for i := 1 to numberofbins do { fill sort dat with zeros }
      dataarray[i] := 0;
    DoDrawGrid;
    DoPlotData;
    DoDrawInfoBar;
  end;

  procedure DoHandleScroll (whichControl: ControlHandle; part: Integer);

    var
      coarse: LongInt;
      coarseMove: Str255;
      delta: LongInt;
      temp: LongInt;
      value: Integer;

  begin

    value := GetCtlValue(whichControl);
    if part <> 0 then
      begin
        if whichControl^^.contrlTitle = 'vScrollBar' then
          case part of
            InUpButton:
              begin
                writegpib('+W');    { clockwise rotation }
                writegpib(Concat('NW', moRes));     {steps/move to moRes }
                writegpib('MW');     { move W axis }
                StringToNum(moRes, delta);
                delta := -delta;
                DoDrawVideo;
              end;
            InDownButton:
              begin
                writegpib('-W');    { counterclockwise rotation }
                writegpib(Concat('NW', moRes));    { steps/move to moRes }
                writegpib('MW');    { move W axis }
                StringToNum(moRes, delta);
                DoDrawVideo;
              end;
            InPageUp:
              begin
                StringToNum(moRes, temp);     { convert moRes to longint }
                coarse := (temp) * 10;        { multiply by 3 }
                NumToString(coarse, coarseMove);  { convert coarse to string }
                writegpib('+W');    { clockwise rotation }
                writegpib(Concat('NW', coarseMove));     {steps/move to moRes }
                writegpib('MW');     { move W axis }
```

```
                StringToNum(moRes, delta);
                delta := -coarse;
              end;
            InPageDown:
              begin
                StringToNum(moRes, temp);     { convert moRes to longint }
                coarse := (temp) * 10;        { multiply by 3 }
                NumToString(coarse, coarseMove);  { convert coarse to string }
                writegpib('-W');    { clockwise rotation }
                writegpib(Concat('NW', coarseMove));     {steps/move to moRes }
                writegpib('MW');     { move W axis }
                StringToNum(moRes, delta);
                delta := coarse;
              end;

            otherwise
          end;

        if whichControl^^.contrlTitle = 'hScrollBar' then
          case part of
            InUpButton:
              begin
                writegpib('-X');    { clockwise rotation }
                writegpib(Concat('NX', horizRes));       {steps/move to moRes }
                writegpib('MX');     { move X axis }
                StringToNum(horizRes, delta);
                delta := -delta;
                DoDrawVideo;
              end;
            InDownButton:
              begin
                writegpib('+X');    { counterclockwise rotation }
                writegpib(Concat('NX', horizRes));      { steps/move to moRes }
                writegpib('MX');    { move X axis }
                StringToNum(horizRes, delta);
                DoDrawVideo;
              end;
            InPageUp:
              begin
                StringToNum(horizRes, temp); { convert horizRes to longint }
                coarse := (temp) * 10;        { multiply by 3 }
                NumToString(coarse, coarseMove);  { convert coarse to string }
                writegpib('-X');    { clockwise rotation }
                writegpib(Concat('NX', coarseMove));     {steps/move to horizRes }
                writegpib('MX');     { move X axis }
                StringToNum(horizRes, delta);
                delta := -coarse;
              end;
            InPageDown:
              begin
                StringToNum(horizRes, temp); { convert horizRes to longint }
                coarse := (temp) * 10;        { multiply by 3 }
                NumToString(coarse, coarseMove);  { convert coarse to string }
                writegpib('-X');    { clockwise rotation }
                writegpib(Concat('NX', coarseMove));     {steps/move to horizRes }
                writegpib('MX');     { move X axis }
                StringToNum(horizRes, delta);
                delta := coarse;
              end;
            otherwise
          end;
      end;
    SetCtlValue(whichControl, GetCtlValue(whichControl) + delta);
```

```
  end;

  procedure DoScrollImage (whichControl: ControlHandle; part: Integer; myPt: Point);

  begin
    if part <> InThumb then
      part := TrackControl(whichControl, myPt, @DoHandleScroll)
    else
      begin
        part := TrackControl(whichControl, myPt, nil);
      end;
  end;    { of procedure DoScrollImage}

  procedure DoImageZoomIn;

  begin
    InvertRect(inRect);
    InvertRect(inRect);
    writegpib('-Y');      { counterclockwise rotation }
    writegpib(Concat('NY', focusRes));  { steps/move to moRes }
    writegpib('MY');      { move Y axis }

  end;

  procedure DoImageZoomOut;

  begin
    InvertRect(inRect);
    InvertRect(inRect);
    writegpib('+Y');      { counterclockwise rotation }
    writegpib(Concat('NY', focusRes));  { steps/move to moRes }
    writegpib('MY');      { move Y axis }
  end;

  procedure DoCenterStages;
    var
      theItem: Integer;
      theControl: ControlHandle;

  begin
    theItem := DoMisc('Move all stages to their nominal center positions?');
    if theItem = 1 then
      begin
        SetCursor(watch);
        theControl := WindowPeek(myWindow)^.controlList;
        while theControl <> nil do     { put scroll bars in center positions }
          begin
            if theControl^^.contrlTitle = 'vScrollBar' then
              SetCtlValue(theControl, 9000);
            if theControl^^.contrlTitle = 'hScrollBar' then
              SetCtlValue(theControl, 10650);
            theControl := theControl^^.nextControl;
          end;
        DoGoHome;
      end;
    SetCursor(arrow);
  end;

  procedure DoMenuDeskew;
{ handles deskew menu command }
    var
      theItem: Integer;
```

```
    begin
      theItem := DoMisc('Deskewing will overwrite existing values. Do you want to
   continue?');
      case theItem of
        1:  { OK button }
          begin
            SetCCursor(coffee);
            calcDeskew;{perform calculated instead of measured, for now}
  {deskew;}
          end;
        2:  { cancel button }
          begin
          end;
      end;  { case }
    end;

    procedure DoObjectInfo;
  { brings up dialog with object information }

      var
        box: Rect;
        item: Handle;
        itemType: Integer;
        result: Integer;
        savePort: GrafPtr;
        theDialog: DialogPtr;
        theItem: Integer;

    begin
      GetPort(savePort);
      theDialog := GetNewDialog(ObjectID, nil, Pointer(-1)); { get dialog box }
      DoCenterDialog(theDialog);
      DoOutline(theDialog);
      GetDItem(theDialog, 3, itemType, item, box);
      SetIText(item, objectName);
      SelIText(theDialog, 3, 0, 32767);

      GetDItem(theDialog, 4, itemType, item, box);
      SetIText(item, RAValue);

      GetDItem(theDialog, 5, itemType, item, box);
      SetIText(item, decValue);

      GetDItem(theDialog, 6, itemType, item, box);
      SetIText(item, observatory);
      ModalDialog(nil, theItem);              { put dialog box up; get result }
      case theItem of
        1:  { OK Button }
          begin
            GetDItem(theDialog, 3, itemType, item, box);
            GetIText(item, objectName);
            SetWTitle(myWindow, objectName);
            GetDItem(theDialog, 4, itemType, item, box);
            GetIText(item, RAValue);
            GetDItem(theDialog, 5, itemType, item, box);
            GetIText(item, decValue);
            GetDItem(theDialog, 6, itemType, item, box);
            GetIText(item, observatory);
          end;
        2:  { cancel button }
          begin
          end;
      end;  { case }
```

```
    DisposDialog(theDialog);            { get rid of dialog box }
    SetPort(savePort);
    DoDrawInfoBar;
  end;

  procedure DoGPIBCommand;
{ sends any command to GPIB board }

    var
      box: Rect;
      item: Handle;
      itemType: Integer;
      command: Str255;
      savePort: GrafPtr;
      theDialog: DialogPtr;

  begin
    GetPort(savePort);
    theDialog := GetNewDialog(CommandID, nil, Pointer(-1)); { get dialog box }
    DoCenterDialog(theDialog);
    DoOutline(theDialog);
    ParamText('Send GPIB command:', '', '', '');
    repeat
      SelIText(theDialog, 3, 0, 32767);
      ModalDialog(nil, theItem);   { put dialog box up; get result  }
      GetDItem(theDialog, 3, itemType, item, box);
      GetIText(item, command);      { get command }
      case theItem of
        1:
          begin
            writeGPIB(command);
          end;
        2:
          begin
          end;
      end;
    until theItem = 2;
    DisposDialog(theDialog);  { get rid of dialog box }
    SetPort(savePort);
  end;

  procedure DoSerialCommand;
{ sends any command to serial port }

    var
      box: Rect;
      item: Handle;
      itemType: Integer;
      command: Str255;
      savePort: GrafPtr;
      theDialog: DialogPtr;

  begin
    GetPort(savePort);
    theDialog := GetNewDialog(CommandID, nil, Pointer(-1)); { get dialog box }
    DoCenterDialog(theDialog);
    DoOutline(theDialog);
    ParamText('Send Serial command:', '', '', '');
    repeat
      ModalDialog(nil, theItem);   { put dialog box up; get result  }
      GetDItem(theDialog, 3, itemType, item, box);
      GetIText(item, command);      { get command }
      case theItem of
```

```
        1:
          serialWrite(command);
        2:
          begin
          end;
      end;
    until theItem = 2;
    DisposDialog(theDialog);  { get rid of dialog box }
    SetPort(savePort);
  end;

  procedure DoDrawStatusWindow;
    var
      i: Integer;
      statusFrame: Rect;
      grid: Integer;
      offset: integer;
  begin
    grid := 15;
    offset := 30;
    RGBForeColor(blackRGB);
    with myWindow^.portRect do
      SetRect(statusFrame, right - ((8 * grid) + offset), bottom - ((8 * grid) + offset
right - offset, bottom - offset);
    RGBForeColor(gridColor);
    with statusFrame do
      begin
        for i := 1 to 8 do
          begin
            MoveTo(left, top + (grid * i));
            LineTo(right, top + (grid * i));
            MoveTo(left + (grid * i), top);
            LineTo(left + (grid * i), bottom);
          end;
      end;
    RGBForeColor(blackRGB);
    FrameRect(statusFrame);
  end;

  procedure DoEDStatus;
    var
      savePort: GrafPtr;
      width: integer;
      height: integer;
      gridSpacing: Integer;

  begin
    GetPort(savePort);
    gridSpacing := 15;
    statusWindow := GetNewCWindow(windoid, nil, pointer(-1));
    DoDrawStatusWindow;
    SetWTitle(statusWindow, 'Status');
    with screenBits.bounds do          { create full screen window }
      begin
        width := 8 * gridSpacing;
        height := 8 * gridSpacing;
        SizeWindow(statusWindow, width, height, TRUE);
        MoveWindow(statusWindow, 100, 100, TRUE);
        ShowWindow(statusWindow);
      end;
    SetPort(myWindow);
  end;
```

```
    procedure DoTextWindow;
    begin
      if not textisvisible then
        begin
          showtext;
          SetItem(GetMHandle(UtilityMenu), 8, 'Hide Text');
        end;
      if textisvisible then
        begin
          hideAll;
          SetItem(GetMHandle(UtilityMenu), 8, 'Show Text');
        end;
    end;


end.
```

THIS PAGE INTENTIONALLY LEFT BLANK

```
program DigitalProfiler;

{ July 17,1991: Began project  RS }
{ July 18, 1991: Added in ML's code  RS & ML }
{ July 19, 1991: Deleted vestiges of word processor code RS }
{  07/22/91 added uses dpgpib. added opengpib to initialize}
{           added select MO code}
{ July 29,1991: Commented out default MO selection in init routine }
{           enabled deskew menu item, no dialog box yet}
{  8/7/91  think i fixed the NIL dereference problem in histogram by }
{          doing a new(mywindow) first thing in initialize}
{  9/3/91  changed mo mapping EM }
{ Things to do: }
{ Add limits for horizontal stage to keep from hitting can}
{ figure out how to close MicroTV driver }
{ Raw data window}
{ added center stages routine to keep the MOs from losing their minds}

{$R+}

  uses
    DPGlobals, DPDialogs, instrintf, {interface to the instrument hardware}
    dpgpib, {interface to GPIB bus}
    SANE, DPFile, DPUtilities, DPHistogram, DPVideo;

  procedure DoSizeScrollBars (whichWindow: WindowPtr);
{ ••• Procedure DoSizeScrollBars resizes scroll bars ••• }

    var
      theControl: ControlHandle;

  begin
    theControl := WindowPeek(whichWindow)^.controlList;
    while theControl <> nil do
      begin
        with whichWindow^.portRect do
          begin
            if theControl^^.contrlTitle = 'vScrollBar' then
              begin
                HideControl(theControl);
                SizeControl(theControl, 16, 110);
                MoveControl(theControl, right - 25, 29);
                ShowControl(theControl);
              end;
            if theControl^^.contrlTitle = 'hScrollBar' then
              begin
                HideControl(theControl);
                SizeControl(theControl, 130, 16);
                MoveControl(theControl, right - 26 - 128, 30 + 108);
                ShowControl(theControl);
              end;
          end;
        theControl := theControl^^.nextControl;
      end;
  end;

  procedure CursorAdjust;
{ set cursor depending on mouseLocation }
    var
      mousePt: point;

  begin
    GetMouse(mousePt);
```

```
    SetCursor(arrow);
  end; { Procedure CursorAdjust }

  procedure DoAbout;
{ ••• Procedure DoAbout displays About Digital Profiler... dialog ••• }
    var
      amount: Str255;
      free: LongInt;
      savePort: GrafPtr;
      tempRect: Rect;
      theItem: Integer;
      theDialog: DialogPtr;

  begin
    GetPort(savePort);
    SetCCursor(logo);
    free := FreeMem div 1000;
    NumToString(free, amount);
    ParamText(amount, '', '', '');
    theDialog := GetNewDialog(AboutID, nil, Pointer(-1));    { get dialog box }
    DoCenterDialog(theDialog);
    DoOutline(theDialog);
    TextFont(Geneva);
    TextSize(12);

    ModalDialog(nil, theItem);              { put dialog box up; get result  }
    DisposDialog(theDialog);               { get rid of dialog box }
    TextFace([]);
    SetPort(savePort);
    SetCursor(arrow);
  end; { of DoAbout }

  procedure DoNewWindow;
{ Procedure DoNewWindow initializes the editing window }

    var
      height: Integer;
      hScroll: ControlHandle;
      newRect: Rect;
      number: Integer;
      temp: Integer;
      temporary: Str255;
      vScroll: ControlHandle;
      width: Integer;

  begin
    myWindow := GetNewCWindow(mainWindow, nil, Pointer(-1));
    vScroll := GetNewControl(1000, myWindow);
    SetCtlValue(vScroll, 9000);
    hScroll := GetNewControl(1001, myWindow);
    SetCtlValue(hScroll, 10650);
    SetPort(myWindow);
    SetWTitle(myWindow, fileName);
    with screenBits.bounds do          { create full screen window }
      begin
        width := screenBits.bounds.right - screenBits.bounds.left - 8;
        height := screenBits.bounds.bottom - screenBits.bounds.top - menuHeight - 4;
        SizeWindow(myWindow, width, height, TRUE);
        MoveWindow(myWindow, screenBits.bounds.left + 4, screenBits.bounds.top + menuHe
TRUE);
        DoSizeScrollBars(myWindow);
        ShowWindow(myWindow);
      end;
```

```
    wInfo := WInfoHandle(NewHandle(SizeOf(windowInfo)));
    SetWRefCon(myWindow, ORD(wInfo));
    HLock(handle(wInfo));
    with wInfo^^ do
      begin
{fill in window record }
      end;
  end; { Procedure SetUpWindows }

  procedure DoMenuSelect (codeword: longint);
{ Procedure DoMenuSelect handles any pulldown menu events that occur }

    var
      item: Integer;      { result from Save Changes dialog }
      refNum: Integer;    { desk accesory reference number }
      theDA: Str255;      { desk accesory name }
      theMenu: Integer;   { menu selected }
      savePort: GrafPtr;  { current grafPort }

  begin
    theMenu := HiWord(codeword);
    theItem := LoWord(codeword);

    if (theItem > 0) then
      case theMenu of          { cases for the menus }

        AppleMenu:
          if theItem = 1 then
            DoAbout
          else
            begin
              GetPort(savePort);
              SetCursor(Arrow);
              GetItem(GetMHandle(AppleMenu), theItem, theDA);
              refNum := OpenDeskAcc(theDA);
              SetPort(savePort);
            end;

        FileMenu:
          case theItem of
            1:
              SysBeep(10);
            2:
              DoOpenMenu;
            5:        { save menu }
              begin
                if hasFileName then
                  begin
                    DoSaveAFile(fileName, vRefNum);
                  end;
                if not hasFileName then
                  DoSaveFileName;
              end;
            6:        { save changes menu item }
              DoSaveFileName;
            8:        { export graph as PICT file }
              DoSavePICTFile;
            9:        { export text }
              DoSaveTEXTFileName;
            11:       { page setup }
              SysBeep(10);
            12:       { print graph }
              DoPrinting;
```

```
            14:
              begin
                item := DoSaveChanges('quitting');
                if item <> 2 then { 2 is cancel button }
                  done := TRUE;
              end;
          end;

        EditMenu:
          case theItem of
            1:      { undo }
              SysBeep(1);
            3:      { cut }
              SysBeep(1);
            4:      { copy }
              SysBeep(1);
{DoVideoToClipboard;}
            5:      { paste }
              SysBeep(1);
          end;

        DataMenu:
          case theItem of
            1:
              DoAcquireData;
            2:
              DoClearData;
            3:
              DoRealTimeOptions;
            5:
              DoNumberofScans;
            6:
              DoNumberofBins;
            7:
              DoNumberofBoards;
            8:
              DoThreshold;
            9:
              DoEnableChannel;
            11:
              DoPlotData;
            12:
              DoColorPicker;
          end;  { case }

        ImageMenu:
          case theItem of
            3: {start scan mirror}
              startmotor;
            5: {stop scan mirror}
              stopmotor;
            7:
              DoResolution;
            8:
              DoCenterStages;
            10:
              DoCalibrateStages;
            otherwise
              begin
              end;
          end;      { ImageMenu case }

        MOMenu:
```

```
            DoMoMenu(theItem);

          SpeedMenu:
            DoScanSpeed(theItem);

          UtilityMenu:
            case theItem of
              1:
                DoObjectInfo;
              2: {show parameters}
                DoParameters;
              4: {deskew}
                DoMenuDeskew;
              6:
                DoGPIBCommand;
              7:
                DoSerialCommand;
              9:      { show enable/disable status window }
                begin
{DoEDStatus;}
                end;
              10: {show text window}
                ShowText;
            end;

        end;  {case the menu of}
      HiLiteMenu(0);
    end; { Procedure DoMenuSelect }

    procedure DoUpdate;
{  handles window update events from WaitNextEvent RS}

      var
        i, j: Integer;          { loop counters }
        videoRect: Rect;        { location of CCD camera picture}
        savePort: GrafPtr;      { currently active port }
        theControl: ControlHandle;
        whichWindow: WindowPtr;
        tempWInfo: wInfoHandle;

    begin
      GetPort(savePort);    { save current grafPort }
      tempWInfo := wInfo;
      whichWindow := WindowPtr(theEvent.message);      { get window to update }
      if whichWindow = myWindow then      { update procedure for main window }
        begin
          SetPort(myWindow);
          BeginUpdate(whichWindow);      { tell toolbox we're redrawing contents }
          EraseRect(whichWindow^.portrect);
          DrawControls(whichWindow); { draw all window controls }
          with whichWindow^.portRect do
            SetRect(videoRect, right - 26 - 128, 29, right - 24, 30 + 109);
          FrameRect(videoRect);
          DoDrawGrid;    { eventually use CopyBits for speed }
          DoPlotData;
          DoDrawInfoBar;
          PlotCIcon(inputRect, inputCICN);
          with videoRect do    { center zoom in/out rects under picture frame }
            begin
              SetRect(inRect, left + ((right - left) div 2) - 16, bottom + 20, left + ((r
 - left) div 2), bottom + 36);
              PlotIcon(inRect, inICON);
              SetRect(outRect, left + ((right - left) div 2), bottom + 20, left + ((right
```

```
    left) div 2) + 16, bottom + 36);
            PlotIcon(outRect, outICON);
          end;
        with wInfo^^ do { drawing using specific window information }
          begin
          end;
        EndUpdate(whichWindow);     { tell toolbox we're finished }
      end;

    if whichWindow <> FrontWindow then
      begin
        GetPort(savePort);
        BeginUpdate(whichWindow);
        wInfo := WInfoHandle(GetWRefCon(whichWindow));
        with wInfo^^ do
          theControl := WindowPeek(whichWindow)^.controlList;
        while theControl <> nil do
          begin
            HiliteControl(theControl, 255);              { then redraw all controls
}
            theControl := theControl^^.nextControl;
          end;
        EndUpdate(whichWindow);
        SetPort(savePort);
      end;
    wInfo := tempWInfo;
  end;

  procedure DoActivate;
{ Procedure DoActivate handles window activate events RS}

    var
      theControl: ControlHandle;

  begin
    theWindow := WindowPtr(theEvent.Message);

    wInfo := WInfoHandle(GetWRefCon(theWindow));
    if Odd(theEvent.modifiers) then        { ACTIVATE EVENT }
      begin
        SetPort(theWindow);
        theControl := WindowPeek(theWindow)^.controlList;
        while theControl <> nil do
          begin
            HiliteControl(theControl, 0);  { then redraw all controls }
            theControl := theControl^^.nextControl;
          end;
      end;

    if not Odd(theEvent.modifiers) then     { DEACTIVATE EVENT }
      begin
        theControl := WindowPeek(theWindow)^.controlList;
        while theControl <> nil do
          begin
            HiliteControl(theControl, 255);  { then redraw all controls }
            theControl := theControl^^.nextControl;
          end;
      end;
  end;      { Procedure DoActivate }

  procedure DoGrow (whichWindow: WindowPtr; thePt: Point);
{ Procedure DoGrow handles mouse click in grow box  }
```

```
    type
      GrowRec = record
          case Integer of
            0: (
                Result: LongInt
            );
            1: (
                Height, Width: Integer
            )
        end;
    var
      growArea: Rect;
      growInfo: GrowRec;

  begin
    with screenBits.bounds do
      SetRect(growArea, 70, 70, right - 5, bottom - 10);        {    set grow region
    with GrowInfo do
      begin                                                  { if it's our window    }
        Result := GrowWindow(whichWindow, thePt, growArea); {   get amt of growth   }
        SizeWindow(whichWindow, Width, Height, True);         {   resize window       }
        InvalRect(whichWindow^.portRect);                   {   set up for update   }
        DoSizeScrollBars(whichWindow);
      end;
    EraseRect(whichWindow^.PortRect);
    with wInfo^^ do
      begin

      end;
  end; { of proc HandleGrow }

  procedure DoContent (whichWindow: WindowPtr; myPt: Point);
{ ••• Procedure DoContent handles mouse-down in content of window ••• }

    var
      part: Integer;
      whichControl: ControlHandle;
      number: Integer;
      ticks, time: Longint;

  begin
    if whichWindow <> FrontWindow then
      begin
        SelectWindow(whichWindow);
        SetPort(whichWindow);
      end;
    GlobalToLocal(myPt);
    if PtInRect(myPt, inputRect) then  { acquire data icon }
      begin
        PlotCIcon(inputRect, inputHiliteCICN);
        ticks := TickCount;
        repeat
          time := TickCount;
        until time > ticks + 8;
        PlotCIcon(inputRect, inputCICN);
        DoAcquireData;
      end;
    if PtInRect(myPt, inRect) then          { zoom in icon }
      begin
        InvertRect(inRect);
        ticks := TickCount;
        repeat
          time := TickCount;
```

```
        until time > ticks + 8;
        InvertRect(inRect);
        DoImageZoomIn;
      end;
    if PtInRect(myPt, outRect) then      { zoom out icon }
      begin
        InvertRect(outRect);
        ticks := TickCount;
        repeat
          time := TickCount;
        until time > ticks + 8;
        InvertRect(outRect);
        DoImageZoomOut;
      end;
    if FindControl(myPt, whichWindow, whichControl) <> 0 then
      begin
        part := FindControl(myPt, whichWindow, whichControl);
        DoScrollImage(whichControl, part, myPt);
      end;
  end;


  procedure DoZoom (whichWindow: WindowPtr; thePt: Point; partCode: Integer);

  begin
    if TrackBox(whichWindow, thePt, partCode) then
      begin
        EraseRect(whichWindow^.portRect);
        ZoomWindow(whichWindow, partCode, TRUE);
        DoSizeScrollBars(whichWindow);
        with wInfo^^ do
          begin

          end;
        InvalRect(whichWindow^.portRect);
      end;
  end;


  procedure DoMouseDown (theEvent: EventRecord);
{ Procedure DealWithMouseDown processes any mouse events that occur. }

  begin
    code := FindWindow(theEvent.where, whichWindow);
    myPt := theEvent.where;

    case code of
      inMenuBar:
        DoMenuSelect(MenuSelect(theEvent.where));

      inGoAway:
        begin
          if TrackGoAway(whichWindow, theEvent.where) then
            begin
              closewindow(whichWindow);
              done := TRUE;
            end;
        end;

      inSysWindow:
        SystemClick(theEvent, whichWindow);

      inContent:
```

```
        DoContent(whichWindow, myPt);

      inDrag:
        DragWindow(whichWindow, myPt, dragRect);

      inGrow:
        DoGrow(whichWindow, myPt);

      inZoomIn:
        DoZoom(theWindow, myPt, code);

      inZoomOut:
        DoZoom(theWindow, myPt, code);
    end    {of case code}
  end; { Procedure DealWithMouseDown }


  procedure DoKeyDown;
{ handles keyboard events, includin menu selection RS }

    var
      keyPressed: char;

  begin
    keyPressed := char(BAnd(theEvent.message, charcodemask));
    if (BAnd(theEvent.modifiers, cmdkey)) <> 0 then    { Command Key }
      DoMenuSelect(menukey(keyPressed))
    else
      case keyPressed of
        'a':
          SysBeep(10);
        otherwise
          case BSR(BAND(theEvent.message, keycodeMask), 8) of
            upArrow:
              SysBeep(10);
            downArrow:
              SysBeep(10);
            leftArrow:
              SysBeep(10);
            rightArrow:
              SysBeep(10);
          end;                {procedure DoKeyDown}
      end;
  end;


  procedure DoFinder;
{ opens/prints documents from the Finder RS}

    var
      count: Integer;          { number of docs to open/print }
      index: Integer;          { loop counter }
      message: Integer;        { tells whether to open or print }
      reply: AppFile;          { record from Finder about files }
      typeList: SFTypeList;    { file types to open }

  begin
    CountAppFiles(message, count);
    if count = 0 then
      DoNewWindow;
    if count <> 0 then
      begin
        case message of
```

```
          0:   { open document from Finder }
            begin
              if count > 1 then   { only able to open 1 document }
                begin
                  theItem := DoMisc('Sorry, Digital Profiler can only open one document
time.');
                  count := 1;
                end;
              for index := 1 to count do
                begin
                  GetAppFiles(index, reply);
                  if reply.fType = 'PICT' then
                    begin
                      theItem := DoMisc('Sorry, Digital Profiler is unable to open PICT
documents. That file was for export only.');
                      DoNewWindow;
                    end;

                  if reply.fType = 'DIMG' then
                    begin
                      DoOpenAFile(reply.fName, reply.vRefNum);
                      DoNewWindow;
                      DoPlotData;
                      ClrAppFiles(index);
                      SetWTitle(myWindow, reply.fName);
                    end;

                  if reply.fType = 'TEXT' then
                    begin
                      theItem := DoMisc('Sorry, Digital Profiler is unable to open TEXT
documents.');
                      DoNewWindow;
                    end;

                  if reply.fType = 'DDAT' then
                    begin
                      theItem := DoMisc(Concat('The document "', reply.fName, '" cannot
opened. It is for use by Digital Profiler only.'));
                      DoNewWindow;
                    end;
                end;     { for 1 to count do }
            end;         { open document from Finder }
          1:   { print document from Finder }
            begin
              if count > 1 then   { only able to open 1 document }
                begin
                  theItem := DoMisc('Sorry, Digital Profiler can only print one documen
a time.');
                  count := 1;
                end;
              for index := 1 to count do
                begin
                  GetAppFiles(index, reply);
                  DoOpenAFile(reply.fName, reply.vRefNum);
                  DoNewWindow;
                  DoPlotData;
                  DoPrinting;
                  ClrAppFiles(index);
                end;
              done := TRUE;
            end;    { print }
        end;   { case }
      end;     { if count }
```

```
  end; { Procedure DoFinder }

  procedure DoSetUpMenus;
{ initializes menus and menubar at startup RS }
{ change to read menu bar only from resource fork using MBAR resource }

    var
      i: Integer;
      menuList: array[1..8] of MenuHandle;  { holds menu info }

  begin
    menuList[1] := GetMenu(AppleMenu);      { read menus in from resource fork }
    menuList[2] := GetMenu(FileMenu);
    menuList[3] := GetMenu(EditMenu);
    menuList[4] := GetMenu(DataMenu);
    menuList[5] := GetMenu(ImageMenu);
    menuList[6] := GetMenu(UtilityMenu);
    menuList[7] := GetMenu(MOMenu);
    menuList[8] := GetMenu(SpeedMenu);


    AddResMenu(menuList[1], 'DRVR');    { add in Desk Accessories (resource type DRVR)}

    for i := 1 to 6 do              { place menus in menulist }
      InsertMenu(menuList[i], 0);

    InsertMenu(menuList[7], -1);    { -1 for hierarchical menu }
    InsertMenu(menuList[8], -1);
    DrawMenuBar;                    { all done so draw the menu bar }
  end; { Procedure DoSetUpMenus }

  procedure Initialize;
{ Procedure Initialize does any initialization neccessary }

    var
      i: Integer;
      event: EventRecord;

  begin
    new(mywindow);              { allocate space and give pointer a value}
    InitCursor;                 { initalize cursors (system call) }
    FlushEvents(everyevent, 0);  { get rid of previous events }
    wCurs := GetCursor(watchCursor);    { read watch cursor from System }
    watch := wCurs^^;
    SetCursor(watch);
    DoSetUpMenus;               { just what it says }
    numberofScans := 10;        { default number of scans }
    numberofBins := 128;        { default number of bins }
    numboards := 7;       { default number of boards (0-7)}
    threshold := 150;           { default threshold in mV}
    realtimeflag := false;      { dont do realtime display until selected}
    numWindows := 0;            { no windows yet }
    hasFileName := false;       { no data, therefor no file name }
    textisvisible := false;     { text window is hidden }
    totalCounts := 0;           { initial number of counts = 0 }
    fileName := 'New Planet'; { default window/file name }
    objectName := 'New Planet';     { default object name }
    RAValue := '0.00';    { default right ascension value }
    observatory := 'Table Mountain';    { default observatory name }
    decValue := '0.00';         { default declination value }
    magFilter := 'None';        { default mag/filter combination }
    scanSpeedValue := '64 scans/sec';
    for i := 1 to 128 do            { fill data array with zero's }
```

```
        dataArray[i] := 0;
      done := FALSE;              { program finished flag }
      dataSaved := TRUE;          { no new data, therefore nothing to save yet }
      inputCICN := GetCIcon(inputCICNID);      { read CICN from resource fork }
      inputHiliteCICN := GetCIcon(inputHiliteCICNID);      { read CICN from resource fork
      SetRect(inputRect, 50, 380, 82, 412);    { rect for above CICN }
      inICON := GetIcon(inICONID);          { read ICON from resource fork }
      outICON := GetIcon(outICONID);        { read ICON from resource fork }
      coffee := GetCCursor(CCoffeeID);      { read cursor from resource fork }
      logo := GetCCursor(CLogoID);          { read cursor from resouce fork }
      with plotColor do                     { original color for histogram; blue }
        begin
          red := 0;
          green := 0;
          blue := integer(56797);
        end;
      with gridColor do    { gee, maybe it's the color for the plot grid? }
        begin
          red := integer(45687);
          green := integer(45687);
          blue := integer(45687);
        end;
      with blackRGB do     { basic black }
        begin
          red := 0;
          green := 0;
          blue := 0;
        end;
      with whiteRGB do     { plain old white }
        begin
          red := integer(65535);
          green := integer(65535);
          blue := integer(65535);
        end;
      dragRect := screenbits.bounds; {limit for window dragging, won't work with multiple
   monitors }
      InsetRect(dragRect, 25, 25);
      cursorRgn := NewRgn; {  pass an empty region to WNE the first time thru }

      SetRect(wrect, 1, 100, 640, 480);{ define size of Think Pascal text window }
      SetTextRect(wrect);  { rectangle for Think Pascal Text window }
      ShowText;            { show Think Pascal text window }
      DoInitVideo;         { initialize and open MicroTV board }
      OpenGPIB;            { open the GPIB board and klinger device}

  { set up the MO mapping table }
  { menu item 1 is MO number 1}
  { number in [ ] corresponds to menu item number }
  { don't forget to include dividing lines in count }
  { menu item [] = mo number}
      mopos[1] := 3;   { 5x ND0 }
      mopos[2] := 2;   { 5x ND1 }
      mopos[3] := 1;   { 5x ND2 }
      mopos[4] := 6;   { 5x ND3 }
      mopos[6] := 9;   { 10x ND0 }
      mopos[7] := 8;   { 10x ND1 }
      mopos[8] := 7;   { 10x ND2 }
      mopos[10] := 5;  { Mirror }
      mopos[12] := 4;  { Aperture }
      monum := 0;      { default monumber}
      WDefault := '0';
      XDefault := '0';
      YDefault := '0';
```

```
    moRes := '1';          { 1 step/move }
    irRes := '100';        { 100 steps/move }
    focusRes := '100';     { 100 steps/move }
    horizRes := '5';       { 5 steps/move }
    with whichWindow^.portRect do
      SetRect(videoRect, right - 26 - 128, 29, right - 24, 30 + 109);
    initdefaults;  {initialize defaults associated with CAMAC crate}
    initeverything; {initialize CAMAC crate hardware}
    enableallchannels; {enable all tube channels}
    DoGoHome;

    disablechannel(2);
    disablechannel(4);
    disablechannel(5);
    disablechannel(7);
    disablechannel(9);
    disablechannel(11);
    disablechannel(16);
    disablechannel(42);
    disablechannel(43);
    disablechannel(44);
    disablechannel(45);
    disablechannel(46);
    disablechannel(56);
    disablechannel(59);

    DoScanSpeed(3); { default scan speed number }
    HideAll;    { hides all Think Pascal windows, including text window }
    DoFinder;   { open/print document from the Finder }
    SetCursor(arrow);
    InvalRect(myWindow^.portrect); { is this really needed? I don't think so }
  end; { Procedure InitToolBox }

begin { Main program block }

  Initialize;

  repeat
    begin
      CursorAdjust;
      SystemTask;
      dodrawvideo;  { draw video window}
      begin
        MoveTo(543, 29);
        LineTo(543, 139);
        MoveTo(478, 84);
        LineTo(608, 84);
      end;
      gotEvent := WaitNextEvent(everyEvent, theEvent, 30, cursorRgn);
      if gotEvent then
        begin
          case theEvent.what of
            mouseDown:
              begin
                realtimeflag := false; {clear the flag whenever a mouse event occurs}
                DoMouseDown(theEvent);
              end;
            updateEvt:
              begin
                DoUpdate;
              end;
            keydown, autokey:
              DoKeyDown;
```

```
            activateEvt:
              DoActivate;
            otherwise
              begin    { do nothing on other events}
              end;
          end; { of event case }
        end; {gotevent}
      {here is the code that is executed each time through the loop}
      if realtimeflag then
        DoRealTime;
    end; {repeat compound statement}
  until done;
  endprogram; {stop the motor}
  DoCloseVideo;
end. { Main program block }
```

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle
Digital Image Profilers for Faint Sources Which have Bright Companions - Phase II SBIR - Final Report

5. Report Date
May 27, 1992

6. Performing Organization Code

7. Author(s)
Elena Morris
Graham Flint
Robert Slavey

8. Performing Organization Report No.
JPL(MMYY)M(NN)

10. Work Unit No.
10-52049

9. Performing Organization Name and Address
Laser Power Corporation
12777 High Bluff Drive
San Diego, CA 92130

11. Contract or Grant No.
NAS7-1103

13. Type of Report and Period Covered
Final Report

12. Sponsoring Agency Name and Address
NASA-JPL
4800 Oak Grove Drive
Pasadena, CA 91109

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

For this program, an image profiling system was developed which offers the potential for detecting extremely faint optical sources that are located in close proximity to bright companions. The approach employed is novel in three respects. First, it does not require an optical system wherein extraordinary measures must be taken to minimize diffraction and scatter. Second, it does not require detectors possessing either extreme uniformity in sensitivity or extreme temporal stability. Finally, the system can readily be calibrated, or nulled, in space by testing against an unresolved singular stellar source. Experimental data derived from laboratory testing of the Phase I breadboard indicate that image profilers of this type will exhibit a performance which approaches the theoretical limit imposed by photon statistics. Specifically, when used in conjunction with a space based telescope in the 2-3 meter class, a multichannel version of the system should permit the detection of 17th and 19th magnitude stellar objects having angular displacements of 0.04 and 0.2 arc seconds, respectively, from a 6th magnitude object. The scope of the Phase II program included the design, fabrication, and laboratory testing of a multichannel instrument, together with field tests performed in conjunction with a 24" telescope at Table Mt.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
	Unclassified/Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified		N/A

PREPARATION OF THE REPORT DOCUMENTATION PAGE

The last page of a report facing the third cover is the Report Documentation Page, RDP. Information presented on this page is used in announcing and cataloging reports as well as preparing the cover and title page. Thus it is important that the information be correct. Instructions for filling in each block of the form are as follows:

Block 1. Report No. NASA report series number, if preassigned.

Block 2. Government Accession No. Leave blank.

Block 3. Recipient's Catalog No. Reserved for use by each report recipient.

Block 4. Title and Subtitle. Typed in caps and lower case with dash or period separating subtitle from title.

Block 5. Report Date. Approximate month and year the report will be published.

Block 6. Performing Organization Code. Leave blank.

Block 7. Author(s). Provide full names exactly as they are to appear on the title page. If applicable, the word editor should follow a name.

Block 8. Performing Organization Report No. NASA installation report control number and, if desired, the non-NASA performing organization report control number.

Block 9. Performing Organization Name and Address. Provide affiliation (NASA program office, NASA installation, or contractor name) of authors.

Block 10. Work Unit No. Provide Research and Technology Objectives and Plans (RTOP) number.

Block 11. Contract or Grant No. Provide when applicable.

Block 12. Sponsoring Agency Name and Address. National Aeronautics and Space Administration, Washington, D.C. 20546-0001. If contractor report, add NASA installation or HQ program office.

Block 13. Type of Report and Period Covered. NASA formal report series; for Contractor Report also list type (interim, final) and period covered when applicable.

Block 14. Sponsoring Agency Code. Leave blank.

Block 15. Supplementary Notes. Information not included elsewhere: affiliation of authors if additional space is required for block 9, notice of work sponsored by another agency, monitor of contract, information about supplements (film, data tapes, etc.), meeting site and date for presented papers, journal to which an article has been submitted, note of a report made from a thesis, appendix by author other than shown in block 7.

Block 16. Abstract. The abstract should be informative rather than descriptive and should state the objectives of the investigation, the methods employed (e.g., simulation, experiment, or remote sensing), the results obtained, and the conclusions reached.

Block 17. Key Words. Identifying words or phrases to be used in cataloging the report.

Block 18. Distribution Statement. Indicate whether report is available to public or not. If not to be controlled, use "Unclassified-Unlimited." If controlled availability is required, list the category approved on the Document Availability Authorization Form (see NHB 2200.2, Form FF427). Also specify subject category (see "Table of Contents" in a current issue of STAR), in which report is to be distributed.

Block 19. Security Classification (of this report). Self-explanatory.

Block 20. Security Classification (of this page). Self-explanatory.

Block 21. No. of Pages. Count front matter pages beginning with iii, text pages including internal blank pages, and the RDP, but not the title page or the back of the title page.

Block 22. Price Code. If block 18 shows "Unclassified-Unlimited," provide the NTIS price code (see "NTIS Price Schedules" in a current issue of STAR) and at the bottom of the form add either "For sale by the National Technical Information Service, Springfield, VA 22161-2171" or "For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402-0001," whichever is appropriate.